Integración de la bidirecionalidad entre Dexcell y Sentilo para la transmisión de datos energéticos

Autor: David Cortes
Data: 10 mayo 2016
Director: Guillem Corominas
Co-Director: Jorge Daniel Negrete Álvarez
Institución del director: Dexma Sensors S.L
Ponente: Felix Freitag
Titulación: Ingeniería Informática Superior
Centro: Facultat d'Informàtica de Barcelona (FIB)
Universidad: Universitat Politècnica de Catalunya (UPC) BarcelonaTech
Empresa: Dexma Sensors S.L

Resumen

Este proyecto plantea una solución viable para unir dos plataformas SaaS del mundo de la gestión energética.

La gestión energética supone un gran avance para el medioambiente, puesto que tiene como intención reducir el consumo energético y por tanto reducir el impacto del hombre en la naturaleza. Por esta razón, el hecho de unir dos plataformas punteras en este sector es un factor muy interesante para intentar crecer conjuntamente y llegar a mas usuarios que desean participar en este movimiento ecológico.

En la presente memoria se desarrollara un proyecto que a través de diferentes tecnologías permite la transmisión de datos energéticos entre Dexcell Energy Manager y Sentilo de manera simple para permitir a los usuarios finales observar sus consumos energéticos en ambas plataformas .

En un principio se plantea explicar mas profundamente qué son estas dos empresas, y cómo se pretende abordar la labor de conectarlas.

Posteriormente se profundizará en el diseño usado para el desarrollo futuro.

Para finalizar se expondrá con que tecnologías se realizó y que pruebas se hicieron para asegurar la calidad del producto.

Tabla de contenido

1 Introducción

En los últimos años la conciencia ecológica o “verde” ha ido cogiendo fuerza en el mundo empresarial, haciendo de éste un tema primordial para cualquier compañía. Por ello, no es extraño encontrarnos con la aparición de numerosos proyectos gubernamentales o empresas destinados a proporcionar una gestión y control de los consumos energéticos para facilitar el ahorro y la eficiencia energética, aumentando la sostenibilidad.

En el sector de la gestión energética encontramos a DEXMA Sensors S.L, una empresa que proporciona los servicios de gestión, análisis y monitorización a tiempo real de datos energéticos a nivel internacional, actualmente con clientes en mas de 31 países, a través de una plataforma en la nube llamada DEXCell Energy Manager.

Por otro lado, en este mismo sector y, por ahora más orientado al mercado español, encontramos Sentilo, una plataforma impulsada por el Ayuntamiento de Barcelona, con la colaboración de empresas privadas y otras organizaciones, con la finalidad de explotar la información “generada por la ciudad”, como por ejemplo datos de tráfico, seguridad, demografía, salud y energía a través de un conjunto de sensores distribuidos en ella para recolectar toda esta información y mostrarla a través de su plataforma http://www.sentilo.io.

1.1 Sentilo

Si bien Sentilo es un proyecto que ha ido introduciéndose progresivamente en los ayuntamientos de todo el país, sigue sin gozar de muchas características imprescindibles para una plataforma destinada a facilitar el ahorro energético. Básicamente, como hemos mencionado anteriormente, consta de un visor pero carece de un proceso de análisis y herramientas para calcular el consumo económico. Debido a estas deficiencias y tras considerar inviable económicamente abordar el desarrollo pertinente de su plataforma, desde Sentilo decidieron confiar en el actual líder en gestión energética a nivel español y uno de los más importantes a nivel europeo en el campo para mejorar su servicio: DEXMA Sensors S.L.

1.2 Dexma Sensors S.L

Para terminar con esta introducción haremos un breve resumen de Dexma Sensors S.L. Esta empresa fue fundada el año 2007 por un grupo de emprendedores provenientes de la Facultad de Informática de Barcelona, Joan Pinyol, Guillem Corominas y Xavier Orduña con la idea de proveer a las empresas una herramienta potente para monitorizar sus consumos y asi facilitar su análisis y reducción. A los pocos años y viendo su crecimiento exponencial en el mercado español, decidieron abrirse camino en otros mercados, de esta manera tras años de crecimiento e inmersión DEXMA ya se ha visto acogida por treinta y tres países en los cinco continentes.

2 Objetivos del proyecto

Como hemos podido observar en la introducción Sentilo requiere asociarse con Dexma para poder proporcionar a sus usuarios una herramienta de análisis tanto de consumos como de costes, y es aquí donde aparece la necesidad de este proyecto, centrado en permitir la integración bidireccional entre estas dos plataformas, para facilitar a los usuarios de Sentilo enviar y almacenar sus datos energéticos en Dexcell Energy Manager y para poder disfrutar de todas las posibilidades de análisis y gestión; además de permitir a los usuarios de Dexcell Energy Manager subir sus datos a Sentilo para poder visualizarlos en su plataforma.

El interés de DEXMA en la realización del presente proyecto es el de ser integrable con una plataforma de IoT - Internet of Things - de uso en la administración pública, para la gestión de datos en ayuntamientos.

Las sinergias pues son claras, ya que como principales clientes de DEXCell Energy Manager, encontramos la administración pública, donde la tendencia es la utilización de plataformas opensource como Sentilo para almacenar, de forma genérica, todos sus datos generados por los sensores de cualquier tipo, ya no sólo energéticos, si no de demografía, salud, tráfico y todo lo relacionado con las Smart Cities.

Por tanto, el objetivo principal de DEXMA es el de favorecer la integración con la plataforma Sentilo para dar un mejor servicio a sus clientes del sector público. Con esto Dexma pretende conseguir los dos siguientes puntos:

a) Proporcionar una interfaz de visualización específica para consumos energéticos a ayuntamientos que ya tienen datos en Sentilo

b) Usar DEXCell como plataforma de recolección y normalización de datos provenientes de equipos de medida heterogéneos para enviar de forma homogénea datos a Sentilo

Para cumplir con estos objetivos se proponen los siguientes objetivos técnicos:

a) Desarrollo de una aplicación que permita al usuario configurar la exportación de datos energéticos de Dexcell a Sentilo a través de los dispositivos de Dexcell.
b) Desarrollar un concentrador virtual en la plataforma de Dexcell que realice la tarea de importar los datos desde Sentilo a Dexcell de manera automática ocultando al usuario todo el sistema de suscripciones y su complejidad.

3 Motivación personal

A la hora de desarrollar este proyecto, han sido tres los criterios que he seguido para decantarme por la opción realizada.

En primer lugar, la oportunidad de realizar toda una aplicación concurrente, puesto que hasta ahora todas las aplicaciones web que había realizado habían sido siempre sencillas sin necesidad de estructuras complejas, mientras que en esta aplicación por la necesidad de proveer a los clientes de sus datos de manera correcta, constante y sin retrasos, se ha optado por herramientas que nos permitan estar tratando los datos de tantos clientes como se desee de manera simultánea, cosa que me ha permitido trabajar con sistemas de colas, “workers”, suscripciones y callbacks.

En segundo lugar, crear desde cero todo un proyecto era para mí una gran fuente de motivación. Desde la creación de las bases de datos, hasta la puesta en marcha en un servidor Heroku del resultado final me permitiría aprender a realizar todas las labores y así tener una visión amplia de todo el desarrollo de creación y mantenimiento de una aplicación web.

Por último, la posibilidad de experimentar con esta aplicación para poder aprender nuevas tecnologías del lado del cliente como nuevos frameworks, por ejemplo Aurelia, destinado a sustituir en breve a ya algunos con renombre como AngularJS.

Por todo lo descrito anteriormente he encontrado interesante desarrollar una aplicación compuesta por multiples módulos que se comunican por APIs REST y sistemas de colas, que por un lado permite a los usuarios de Sentilo configurar fácilmente sus dispositivos en DEXCell, como ofrecer un servicio constante de comunicación con Sentilo para así obtener los datos deseados de forma automática y guardarlos en las bases de datos de Dexma permitiendo su uso a través de un sistema de colas asíncrono.

4 Análisis de requisitos

A continuación se definirá cual es el propósito de este proyecto y su alcance.

El objetivo de nuestra aplicación es que el usuario pueda gestionar y monitorizar el proceso de migración entre las dos plataformas energéticas; por lo tanto el usuario se encuentra frecuentemente interactuando con la aplicación. Es por este motivo que se ha de enfatizar todos los aspectos que faciliten la interacción entre el usuario y la aplicación, es decir, ha de ser “usable”, puesto que de lo contrario a pesar de poder tener un código bien diseñado y eficiente, si deja de ser usable resultaría inservible.

4.1 Características de los usuarios

Esta aplicación esta destinada a cualquier usuario de Dexcell, por lo tanto no requiere que este tenga conocimientos de informática. Por esta razón la aplicación debe tener una interfaz gráfica amigable y intuitiva, que permita al usuario realizar todas las configuraciones de importación y exportación de manera sencilla.

4.2 Características de la plataforma final

A parte de tener en cuenta las características de los usuarios, hace falta tener en cuenta que tanto la parte de importación como exportación están realizadas para ser soportadas en cualquier navegador web actual. Por tanto para poder permitir esta adaptabilidad en cualquier dispositivo nuestra aplicación debe ser clara, concisa y amigable en cualquier tipo de dispositivo para permitir esta adaptabilidad.

4.3 Funcionalidades

Las funcionalidades esenciales de la aplicación para la operación de exportación ofrecidas al usuario son las siguientes :

- Poder configurar la información básica del usuario
- Poder probar la validez de la información configurada
- Poder crear nuevos componentes en Sentilo automáticamente
- Poder crear nuevos sensores asignados a componentes
- Poder asignar a los sensores la frecuencia de exportación de datos.
- Obtener un listado de dispositivos de Dexcell para asignar a los sensores de Sentilo
- Observar a través de logs el estado interno de nuestros sensores para ver su correcto funcionamiento.
- Observar el listado de componentes creados.

Las funcionalidades de importación son las siguientes:

- Poder configurar la información básica del servidor de Sentilo asociado
- Poder obtener el listado de sensores asociados a esta cuenta
- Poder crear a través de estos sensores y una fórmula introducida por el usuario un nuevo dispositivo en Dexcell
- Poder editar estos nuevos dispositivos, eliminarlos, actualizar su fórmula, crear de nuevos.

Tareas del exportador

Configurar información básica usuario: El usuario deberá introducir antes que nada la información relacionada con el servidor de Sentilo al que quiere exportar los datos, para ello debe dirigirse a la pantalla configuración e insertar una dirección url, un token y un proveedor válidos.

Listar componentes: El usuario al entrar en la aplicación lanzará un proceso que le devolverá el listado de componentes configurados.

Creación de un componente: Se presentará un formulario para rellenar al usuario, donde tendrá que insertar información como el nombre deseado para el componente, una descripción para este, si es que lo desea, el tipo de componente que será, y la localización de Dexcell al que hará referencia.

Creación de un sensor: Una vez se haya rellenado correctamente la información del componente se nos mostrará la opción de agregar un nuevo sensor, para su creación el usuario debe asignarle un nombre, un tipo propio de Sentilo y a que dispositivo de Dexma ira relacionado, con que parámetro de Dexcell y que resolución quiere para este.

Editar componente: Una vez creado el componente y insertado en Sentilo, el usuario podrá entrar en su página de configuración para editar los campos de descripción, tipo y editar los sensores también, los campos de nombre y localizaciones quedaran bloqueados.

Editar sensor: En la página de configuración de componentes, el usuario también podrá editar sus sensores, permitiendo actualizar su tipo, el dispositivo del que recoge los datos y las características de este, parámetro asociado y resolución.

Eliminar componente: El usuario puede elegir eliminar cualquier componente desde el listado de componentes visualizado en la página principal, estos deben ser eliminados en este momento también de la página de Sentilo.

Eliminar sensor: El usuario puede escoger eliminar cualquier sensor de sus componentes, estos deben ser eliminados también de la página de Sentilo.

Visualizar logs: El usuario podrá entrar en una pantalla donde se irán cargando cada cinco segundos los nuevos mensajes de estado de sus componentes y sensores, para que así el usuario pueda observar el correcto funcionamiento de estos, o los errores ocurridos.

Exportación manual de datos: A parte del modo automático que definiremos en siguientes apartados, el usuario posee de un modo lanzar un proceso de exportación de datos para un periodo deseado para los sensores deseados.

Tareas del importador

Creación del concentrador: El usuario deberá introducir la información necesaria para que Dexcell pueda conectarse con un servidor de Sentilo y crear un nuevo concentrador interno. Esta información es el nombre del concentrador, dirección URL de la API de sentilo, un token y proveedor válidos, y una zona horaria.

Creación de dispositivos: Una vez introducida la información del concentrador de manera correcta, el usuario podrá seleccionar sensores de su cuenta de Sentilo y asociarles una formula y un parámetro para ser importados a Dexcell.

Eliminación de dispositivos: Una vez creado un concentrador el usuario podrá entrar en su configuración para eliminar cualquier sensor asociado deseado, cancelando la suscripción de este a Sentilo.

Eliminación del concentrador: En el listado de concentradores habilitados el usuario tiene la opción de eliminar todos aquellos concentradores que ya no desee, cancelando automáticamente la suscripción a los sensores de Sentilo.

5 Especificación

Una vez realizado el análisis de requisitos debemos proceder a explicar con más detalle el funcionamiento de estas operaciones realizadas por el usuario. A pesar que el peso de la aplicación no recae en la interacción entre usuario y maquina se ha creído importante detallar estas interacciones o casos de uso.

5.1 Diagrama de casos de uso

Los casos de uso correspondientes a las funcionalidades especificadas. En donde el actor que interactúa es el usuario.

En la ilustración 1 se muestran los diferentes casos de uso para la exportación de datos desde Dexcell

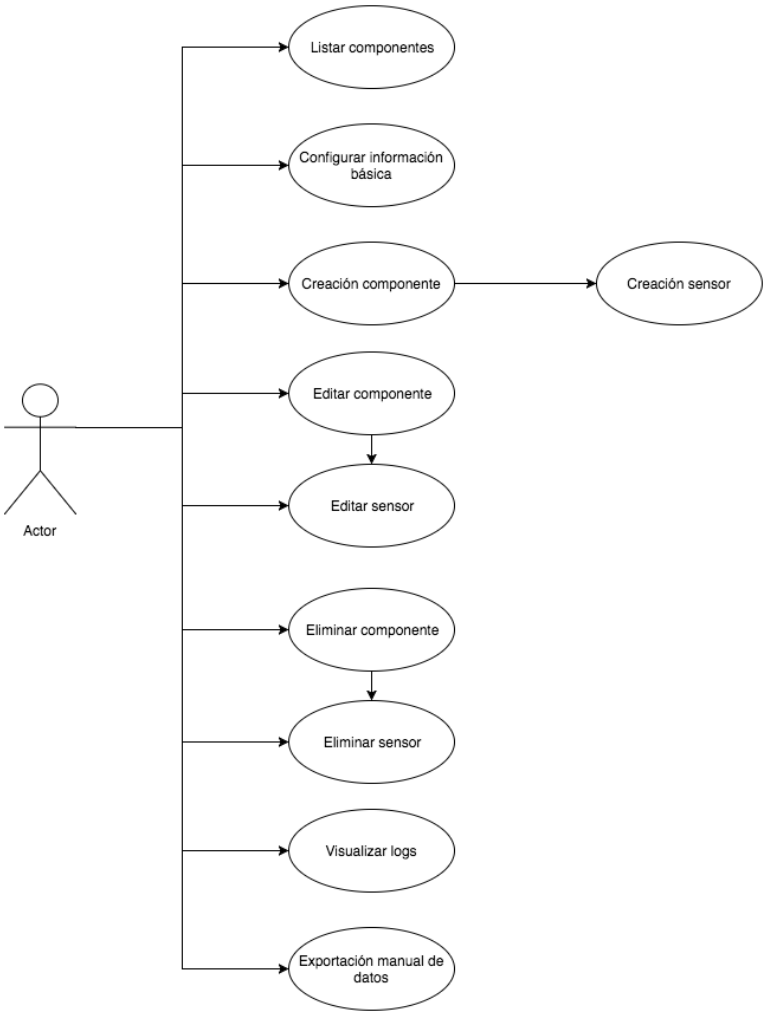


Ilustración 1 Casos de uso exportación

El la ilustración 2 podemos observar los diferentes casos de uso para la importación de datos hacia Dexcell.

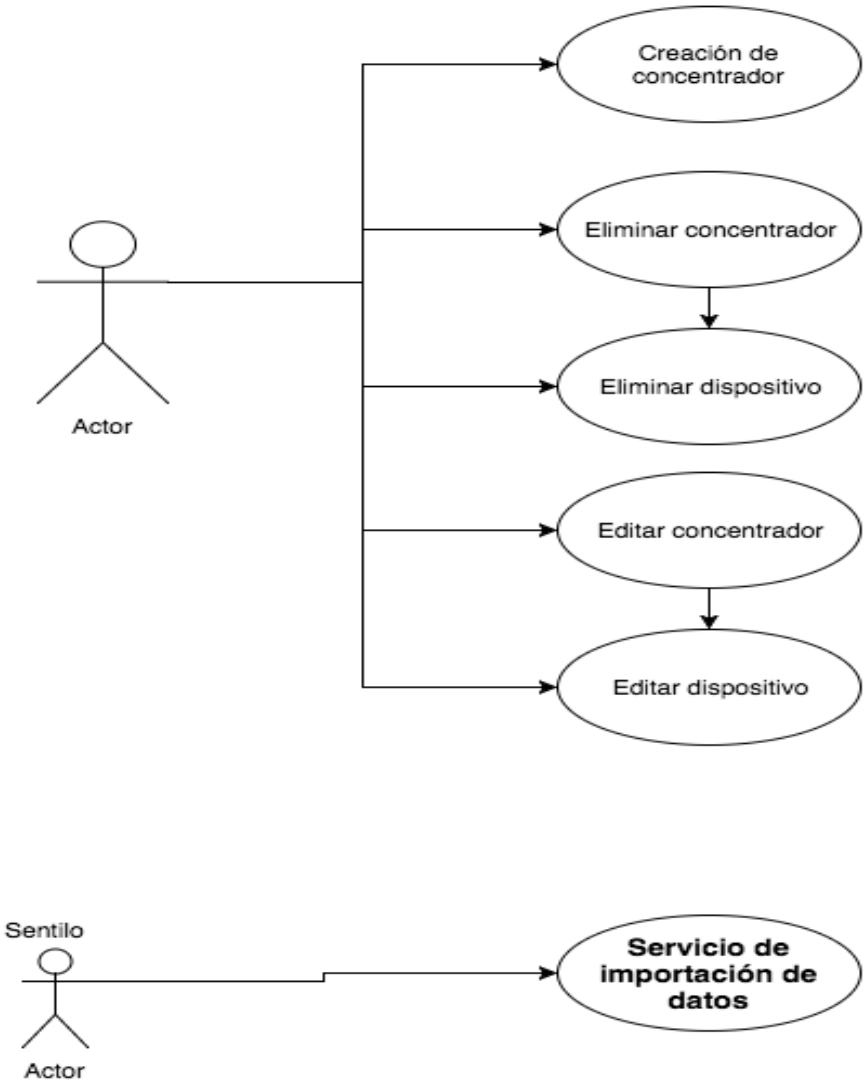


Ilustración 2 casos de uso importación

5.2 Descripción de casos de uso

A continuación se especificarán los diferentes casos de uso expuestos en el apartado anterior. Primero se expondrán los relacionados con la importación y seguidamente los de exportación

5.2.1 Listar componentes

Actor Usuario

Precondiciones Estar registrado en Dexcell y tener la App registrada

Disparador: -

Diagrama de comportamiento

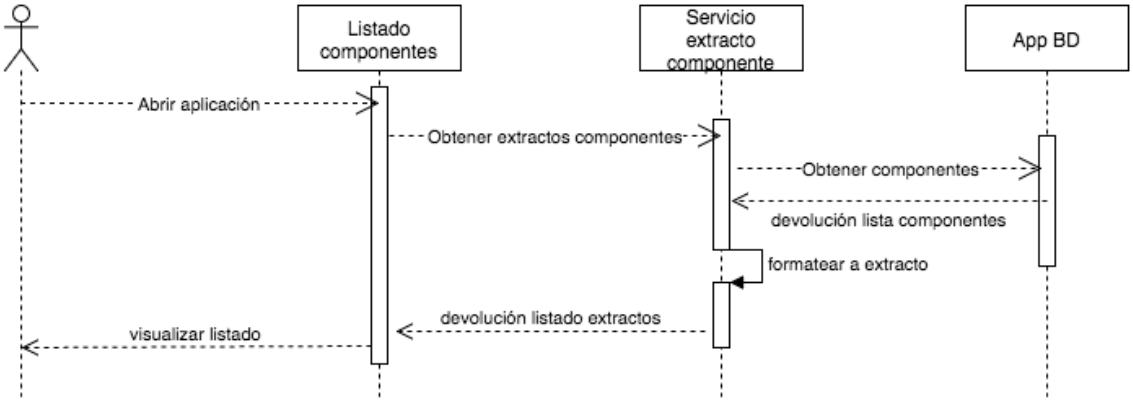


Ilustración 3 Diagrama comportamiento Listar componentes

Escenario principal

1. El usuario abre la aplicación
2. El sistema devuelve la página índice
3. El sistema carga el listado de componentes creados

5.2.2 Configurar información básica

Actor Usuario

Precondiciones Estar registrado en Dexcell y tener la App registrada

Disparador: -

Diagrama de comportamiento

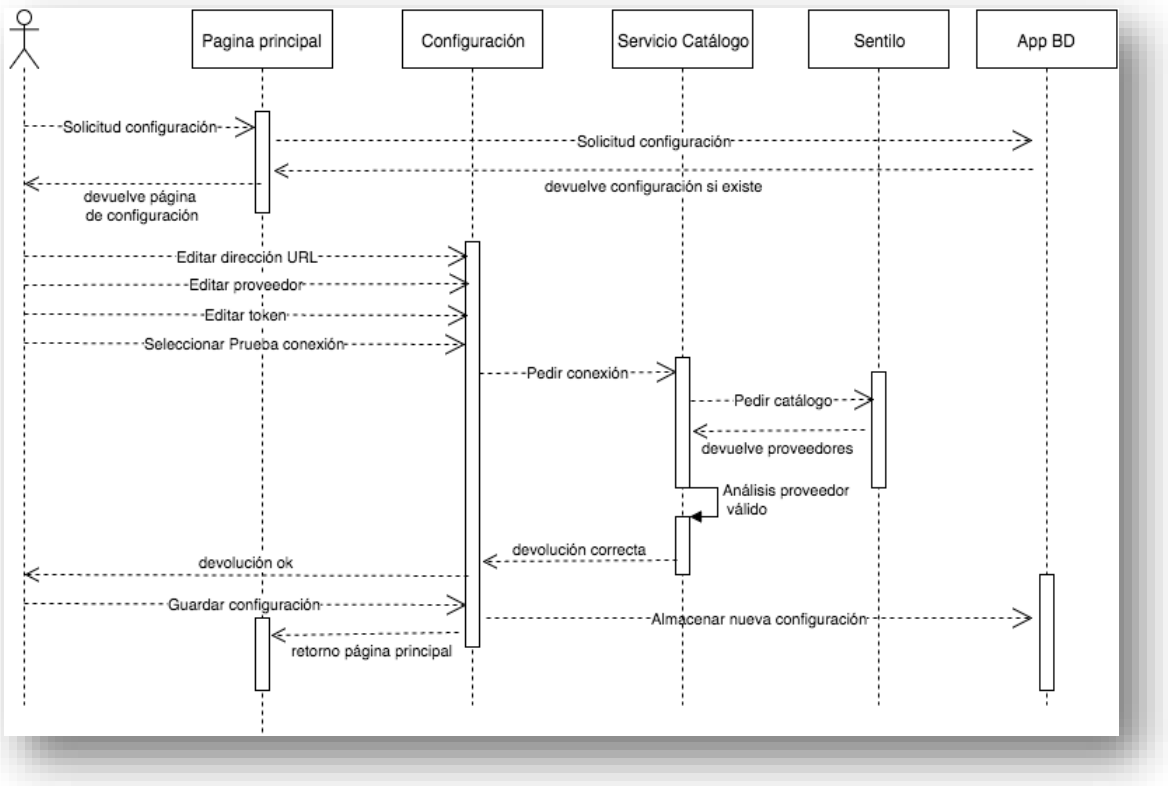


Ilustración 4 Diagrama comportamiento configurar información básica

Escenario principal

1. El usuario hace clic en el botón configuración
2. El sistema muestra una nueva página con la información necesaria para conectarse a Sentilo, es decir hay un campo URL y un campo token.
3. El usuario rellena el campo URL y elige usar el token por defecto de Dexma
4. El usuario clica Conexión de prueba
5. El sistema detecta si la configuración es válida
6. El usuario selecciona guardar
7. El sistema persiste la nueva configuración para la conexión y devuelve un mensaje de guardado correcto al usuario

Escenarios alternativos

6.a El usuario decide que quiere usar su propio proveedor y aprieta en Especificar un proveedor personalizado

6.a.1 El usuario añade un proveedor propio y un token propio.

6.a.2 El sistema notifica al usuario que su configuración es correcta o que es incorrecta.

6.b El usuario decide no aceptar los cambios y hace clic en el botón cancelar

6.b.1 El sistema devuelve al usuario a la página principal.

5.2.3 Creación de un componente

Actor Usuario

Precondiciones La aplicación tiene que estar instalada

Disparador –

Diagrama de comportamiento

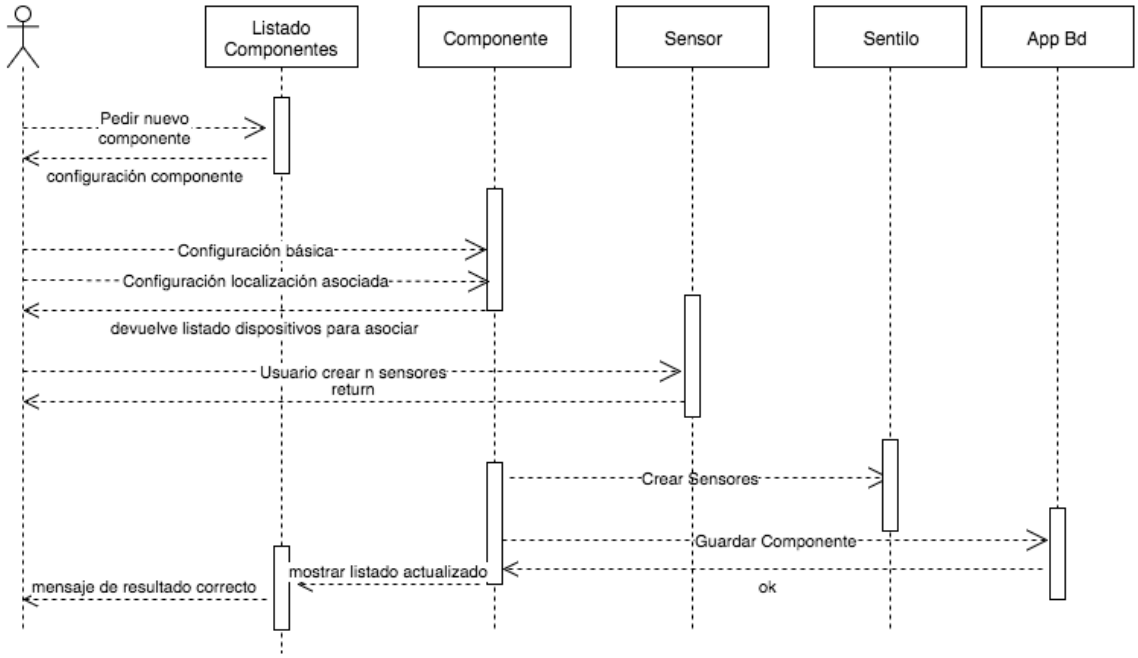


Ilustración 5 Diagrama comportamiento creación componente

Escenario principal

1. El usuario hace clic en el botón nuevo componente.
2. El sistema muestra un formulario con la información necesaria para poder almacenar un componente.
3. El usuario introduce los datos del aparatado información general que son el Id , la descripción, y confirma la acción haciendo clic en guardar
4. El usuario escoge que localización de Dexcell asociará a este componente.
5. El usuario realiza creación de un sensor.
6. El sistema solicita a Sentilo la creación del nuevo componente con sus sensores
7. El sistema almacena los nuevos datos y para terminar redirecciona al usuario al listado de componentes donde se observa el recién creado.

Escenarios alternativos

7.a El sistema deniega la petición del usuario debido a que el atributo Id tiene algún carácter que no pertenece a la codificación ascII.
7.b El usuario hace clic al botón cancelar
7.b.1 El sistema anula todos los valores entrantes y vuelve a la página principal.

5.2.4 Creación de un sensor

Actor Usuario

Precondiciones La aplicación tiene que estar instalada y la información básica del componente debe haber sido rellenada.

Disparador: El usuario se encuentra en el caso de uso creación componente

Diagrama de comportamiento

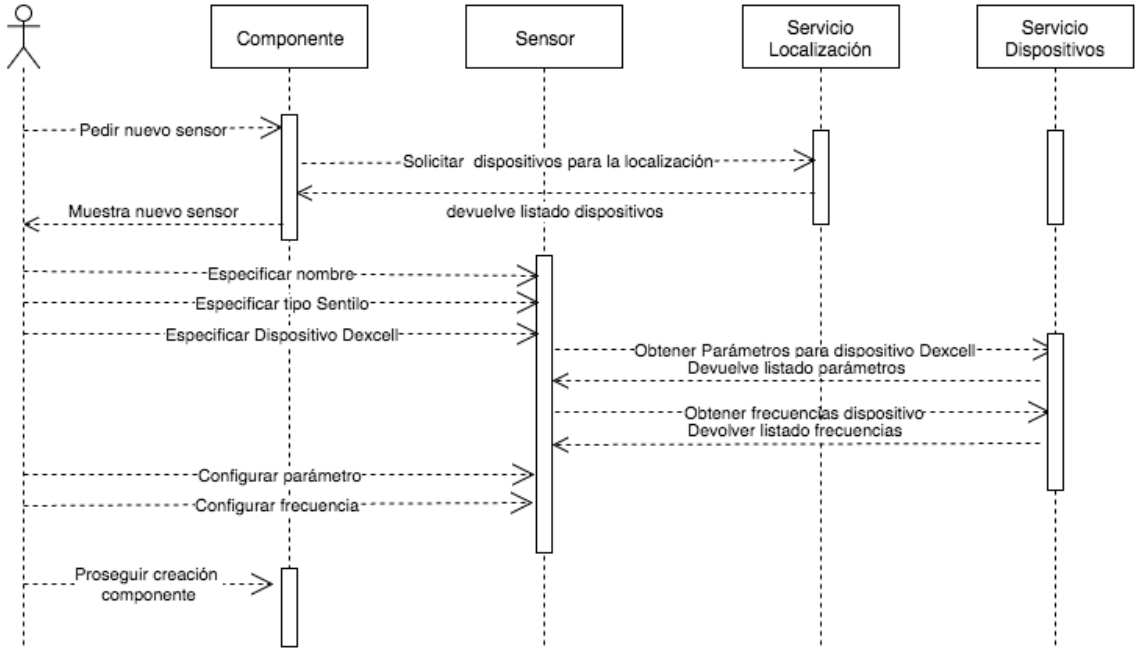


Ilustración 6 Diagrama comportamiento creación sensor

Escenario principal

1. El usuario hace clic en el botón agregar nuevo sensor.
2. El sistema carga y muestra al usuario una nueva línea con la información necesaria a ser rellenada, esta es un campo a llenar donde va el identificador y cuatro desplegables que contienen el tipo del dispositivo en Sentilo, el dispositivo del que provendrán las lecturas desde Dexcell, su parámetro asociado y por último la frecuencia de los datos.
3. El usuario deberá rellenar el campo id y seleccionar un dispositivo de Dexcell.
4. Una vez seleccionado el dispositivo, el sistema buscará los parámetros y frecuencias asociados a este y los mostrará al usuario.
5. El usuario deberá seleccionar uno de los parámetros, una frecuencia y por último proseguir con el caso añadir componente paso 4.

Escenarios alternativos

5.a - El usuario decide que la información del dispositivo es incorrecta y decide eliminarlo apretando el botón con el símbolo de una papelera

5.a.1- El sistema eliminara toda la información referente al sensor mostrado al usuario.

5.b – El usuario decide crear otro sensor y vuelve a clicar el botón agregar nuevo sensor.

5.2.5 Editar componente

Actor Usuario

Precondiciones El componente ya ha sido añadido previamente

Disparador: -

Diagrama de comportamiento

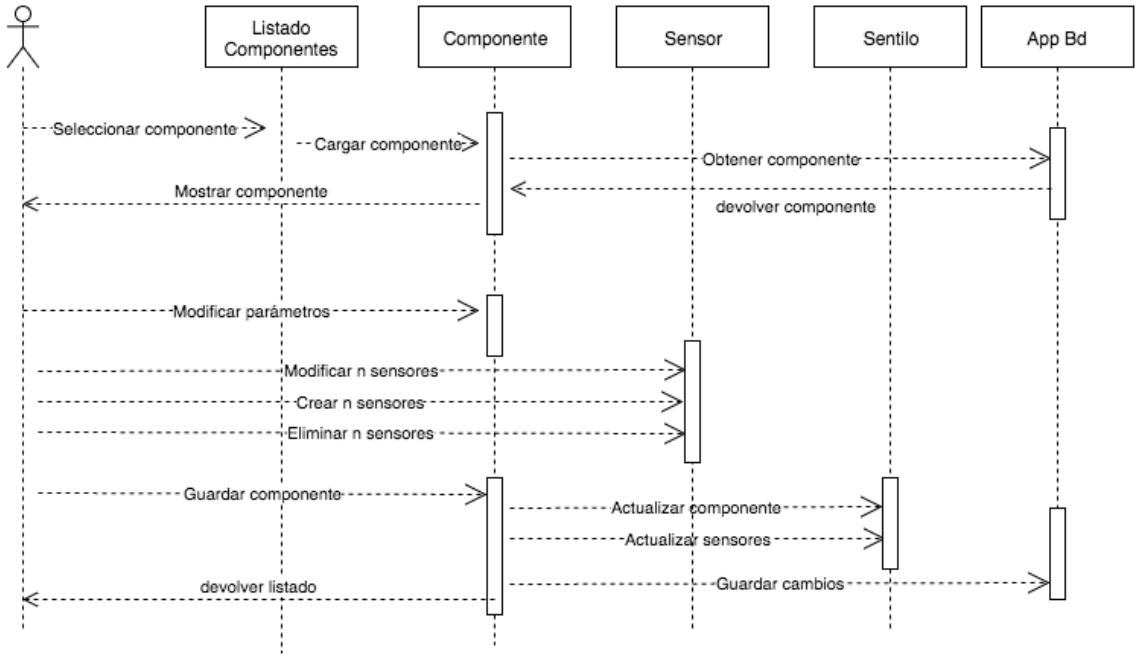


Ilustración 7 Diagrama comportamiento edición componente

Escenario principal

1. El usuario hace clic en el nombre de un componente en el menú principal
2. El sistema muestra al usuario la página de configuración del componente con la información persistida previamente.
3. El usuario podrá modificar cualquier campo del componente menos el identificador de este, que el sistema ha dejado bloqueado y su localización.
4. El usuario podrá editar sensores.
5. Una vez realizadas las modificaciones el usuario ara clic en guardar y el sistema sobrescribirá los datos del componente.

Escenarios alternativos

5.a El usuario decide no realizar los cambios y pulsa en cancelar.

5.a.1 el sistema anulara todos los valores entrantes y vuelve a la página principal.

5.2.6 Eliminar componente

Actor Usuario

Precondiciones el componente ya ha sido añadido previamente

Disparador: -

Diagrama de comportamiento

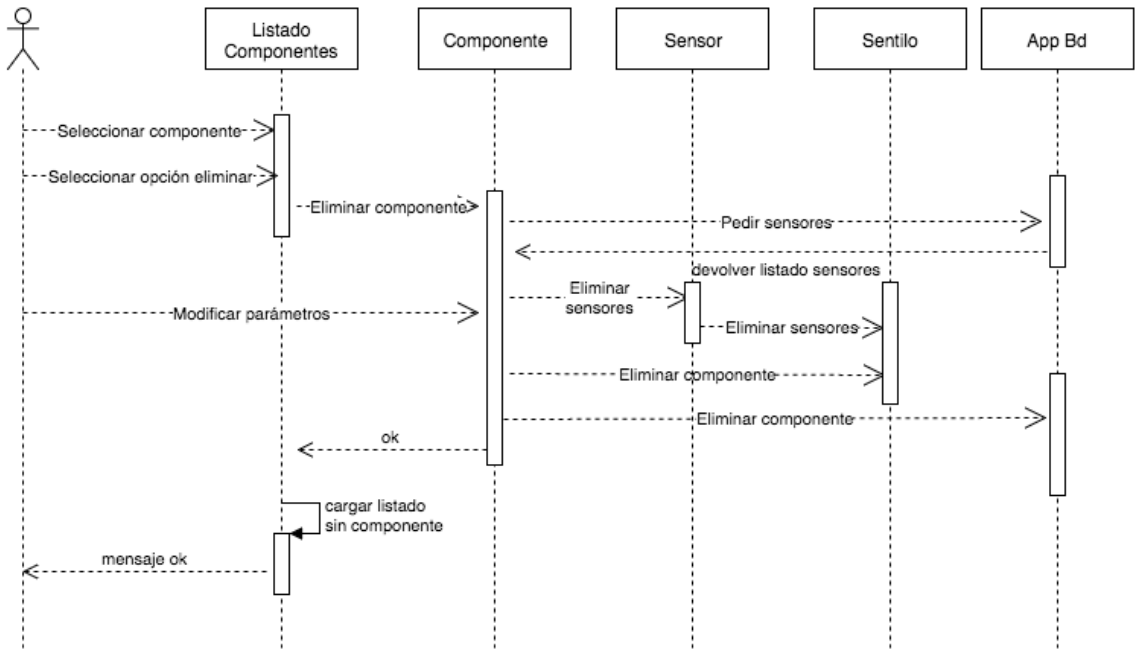


Ilustración 8 Diagrama comportamiento eliminar componente

Escenario principal

1. El usuario hace clic en el checkbox de un componente en el menú principal
2. El sistema desbloquea las acciones del desplegable “acciones”.
3. El usuario selecciona en el desplegable “acciones” eliminar.
4. El sistema elimina de Sentilo el componente y sus sensores
5. El sistema elimina definitivamente el componente y muestra la lista de componentes sin este.

5.2.7 Eliminar sensor

Actor Usuario

Precondiciones el componente ya ha sido añadido previamente con sensores

Disparador: -

Diagrama de comportamiento

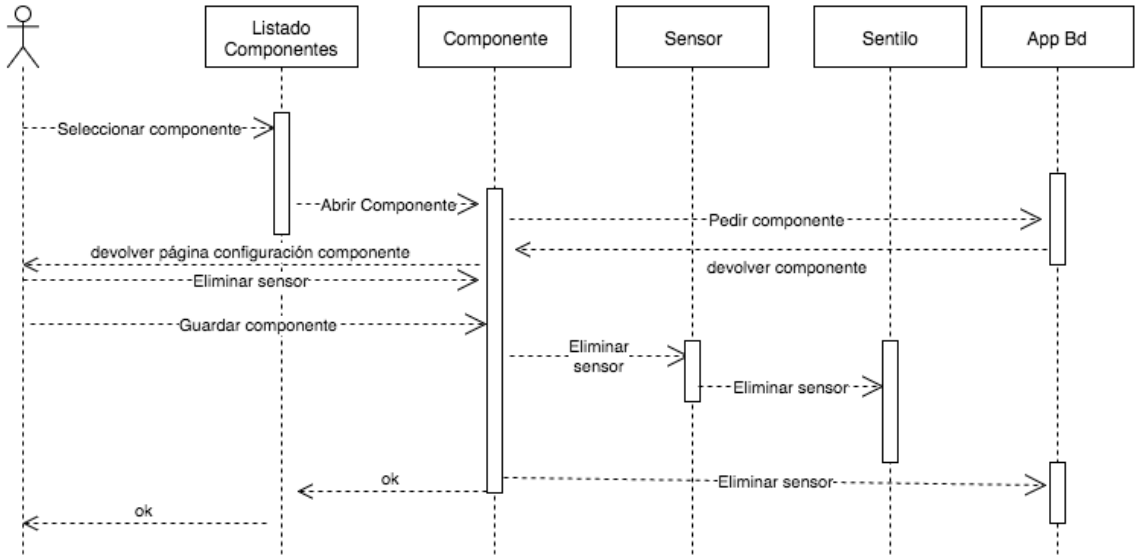


Ilustración 9 Diagrama comportamiento eliminar sensor

Escenario principal

1. El usuario hace clic en un componente en el menú principal.
2. El sistema muestra el componente ya editado.
3. El usuario selecciona en el sensor deseado la opción eliminar.
4. El usuario clica en guardar.
4. El sistema elimina de Sentilo el sensor.
5. El sistema elimina definitivamente el sensor del componente.

5.2.8 Visualizar logs

Actor Usuario

Precondiciones -

Disparador: -

Diagrama de comportamiento

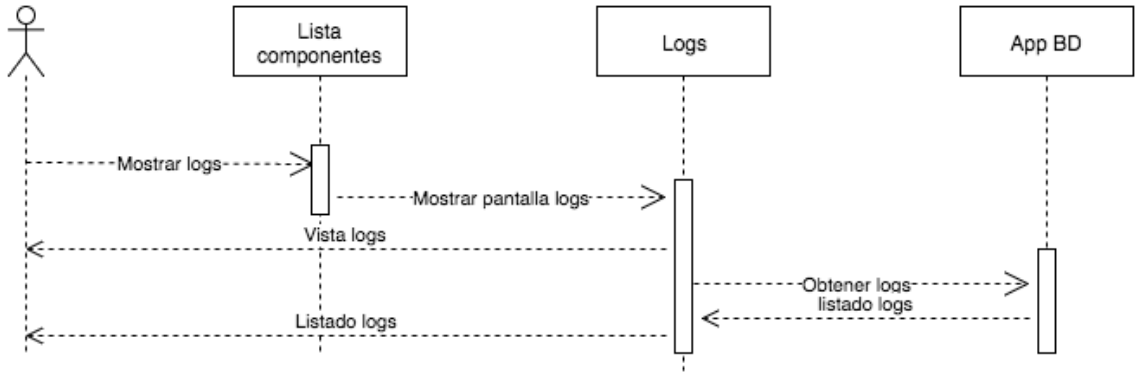


Ilustración 10 Diagrama comportamiento visualizador logs

Escenario principal

1. El usuario quiere ver los resultados de los diversos procesos que realiza internamente la aplicación, así que hace clic en “Logs”.
2. El sistema carga los últimos cien mensajes informativos y los muestra en una nueva página al usuario.
3. El usuario hace clic en volver
4. El sistema devuelve al usuario a la página principal

Escenarios alternativos

3.a El usuario decide actualizar el estado de los logs para ver nuevas actualizaciones y hace clic en actualizar.

3.a.1 El sistema recarga los últimos cien mensajes.

5.2.9 Exportación manual de componentes a Sentilo

Actor Usuario

Precondiciones el componente ya ha sido añadido previamente

Disparador: -

Diagrama de comportamiento

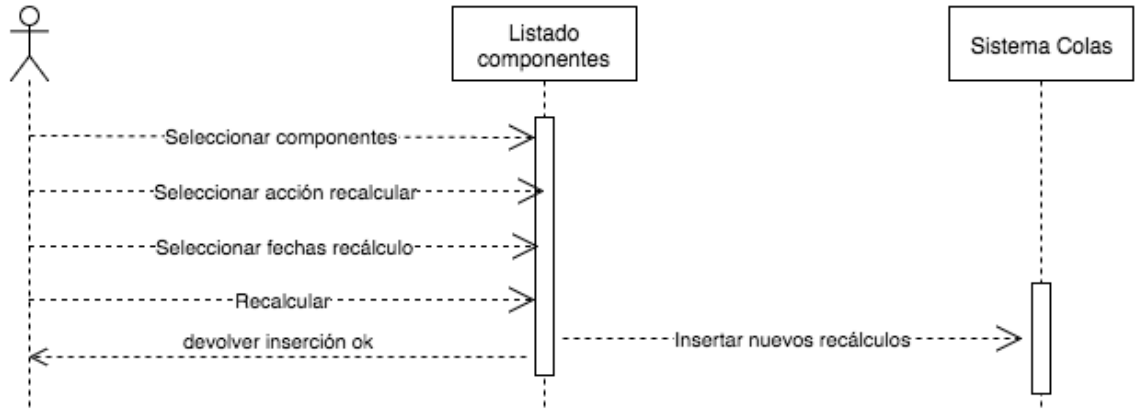


Ilustración 11 diagrama comportamiento exportación manual de componentes

Escenario principal

1. El usuario hace clic en el checkbox de un conjunto de componentes en el menú principal.
2. El sistema desbloquea las acciones del desplegable “acciones”.
3. El usuario selecciona en el desplegable “acciones” la opción “recalcular”.
4. El sistema muestra al usuario un pop up con los nombres de los componentes seleccionados y dos selectores de datas, para introducir la data de inicio de exportación y la de finalización.
5. El usuario rellena los dos selectores y hace clic en “exportar”.
6. El sistema encola la nueva petición, esperando su turno para ser ejecutada.
7. El sistema devuelve al usuario un mensaje para informarle que su petición a quedado almacenada y a continuación cierra el pop up.

Escenarios alternativos

5.a. El usuario decide anular su petición de exportación de datos y hace clic en cancelar.

5.a.1 El sistema cierra el pop up.

5.b El usuario rellena incorrectamente las datas de tal manera que la data de inicio es posterior a la de finalización i hace clic en exportar.

5.b.1 El sistema no deja proseguir la acción y avisa al usuario de su error.

5.c El usuario se olvida de rellenar uno de los dos selectores de datas y hace clic en aceptar.

5.c.1 El sistema no deja proseguir la acción y avisa al usuario de su error y resalta el selector vacío.

5.2.10 Creación del concentrador

Actor Usuario

Precondiciones El usuario debe estar registrado en Dexcell

Disparador -

Diagrama de comportamiento

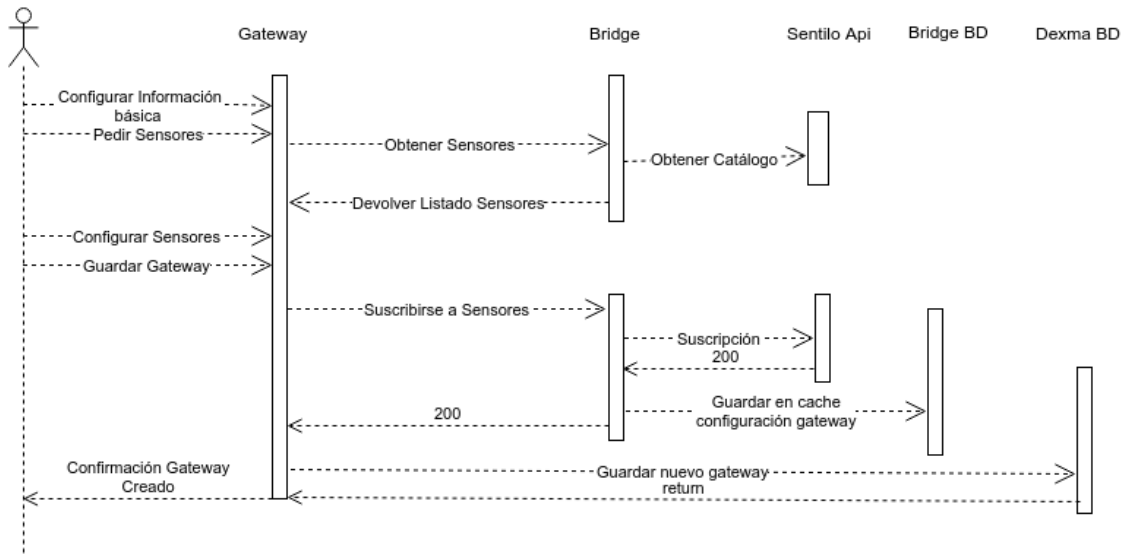


Ilustración 12 Diagrama comportamiento creación del concentrador

Escenario Principal

El usuario decide crear un nuevo concentrador para importar datos de Sentilo a Dexcell. Para ello una vez dentro de nuestra plataforma, el usuario se dirigirá al apartado de configuración de su cuenta y en el subapartado concentradores escogerá crear un nuevo concentrador del tipo “Sentilo Gateway”, una vez dentro podrá observar una pantalla de configuración en la que empezará el proceso definido en la figura 11.

En un primer momento el usuario debe configurar una información básica, que sería:

- el nombre de su concentrador.
- La dirección URL de la API de Sentilo desde donde importar los datos
- Un token de autorización para la URL especificada previamente.
- El proveedor usado en Sentilo
- Su zona horaria

Una vez insertada la información básica el usuario podrá pedir a Dexcell que le muestre el listado de sensores disponibles en su cuenta de Sentilo.

El usuario procederá a escoger los sensores que desee, asignarle un parámetro de conversión (hacia un parámetro válido en Dexcell) y incluso podrà incorporar un cálculo a realizar con este.

Una vez finalizado el proceso de configurar el concentrador, se procederá a su guardado, para esto Dexcell enviará la información al Bridge realizado que procederá a suscribir todos los sensores deseados a traves de l’api de Sentilo.

Para finalizar una vez el Bridge haya suscrito todos los sensores almacenará la configuración del nuevo concentrador y informará a Dexcell de su éxito, quien a su vez notificará al usuario del éxito del proceso a través de un pop-up.

Escenarios alternativos

El usuario decide cancelar la cancelación de creación del concentrador y clica en cualquier momento al botón cancelar.

El sistema redirige al usuario al listado de concentradores.

5.2.11 Eliminación del concentrador

Actor Usuario

Precondiciones El usuario debe haber creado previamente el concentrador en Dexcell

Disparador -

Diagrama de comportamiento

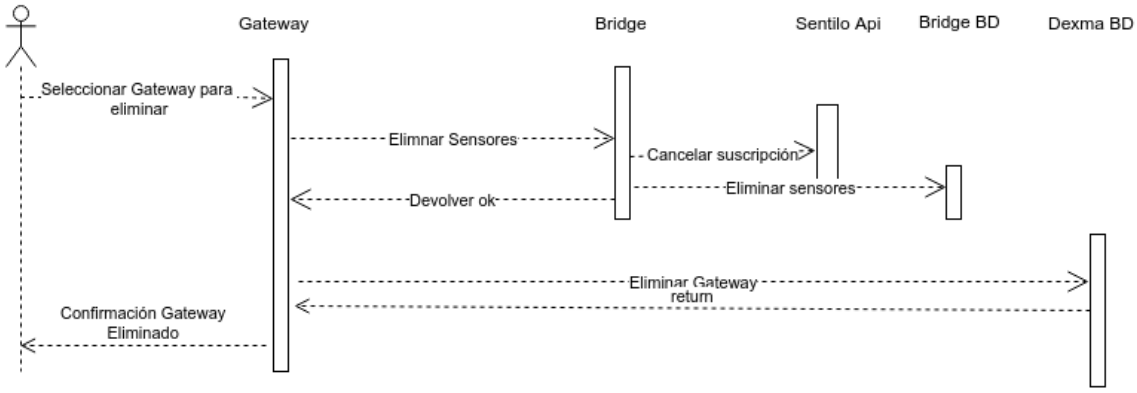


Ilustración 13 Diagrama comportamiento eliminación concentrador

Escenario Principal

El usuario decide eliminar el concentrador para dejar de recibir datos de Sentilo a Dexcell, para ello el usuario en su lista de concentradores configurados, clica el botón de eliminar.

Una vez apretado el sistema activa todo un proceso interno en Dexcell quien llamará al bridge para que se ocupe de cancelar todas las suscripciones del concentrador y que elimine de su base de datos toda información relacionada con el concentrador.

Cuando el bridge termina el sistema informa a Dexcell quien elimina toda información de configuración del concentrador y avisa al usuario del éxito del proceso.

5.2.12 Eliminar dispositivo

Actor Usuario

Precondiciones El usuario debe haber creado previamente el concentrador en Dexcell con sensores asociados

Disparador El usuario se encuentra en el caso editar concentrador o eliminar concentrador.

Diagrama de comportamiento

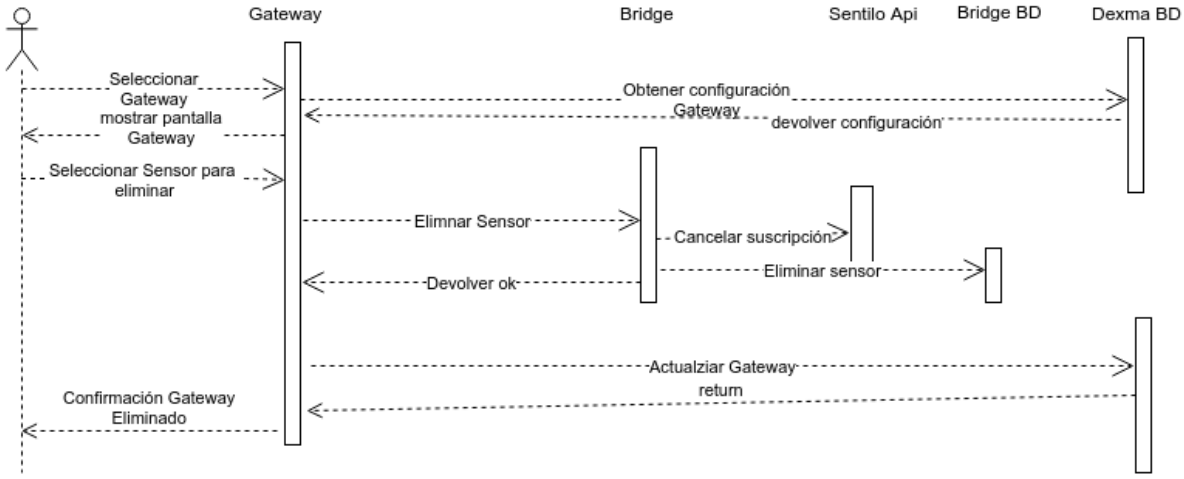


Ilustración 14 Diagrama comportamiento eliminación dispositivo

Escenario Principal

El usuario decide eliminar un sensor de un concentrador, para ello entra en la configuración del concentrador deseado

El sistema solicita a Dexcell la información del concentrador deseado y devuelve al usuario la configuración de este.

El usuario clica el botón eliminar del sensor deseado.

El usuario guarda los cambios del concentrador. A partir de aquí Dexcell internamente sigue unos pasos muy similares a cuando elimina un concentrador.

El sistema llama a nuestro bridge quien cancela la suscripción para ese sensor específico y elimina toda información de configuración sobre este.

Al terminar el bridge el sistema confirma a Dexcell quien también elimina toda información relacionada con el sensor y informa al usuario del éxito del proceso.

Escenario alternativo

El usuario decide no realizar los cambios y abandona la pantalla dándole a cancelar.
El sistema muestra el listado de concentradores disponibles

5.2.13 Editar concentrador

Actor Usuario

Precondiciones El usuario debe haber creado previamente el concentrador en Dexcell con sensores asociados

Disparador -

Diagrama de comportamiento

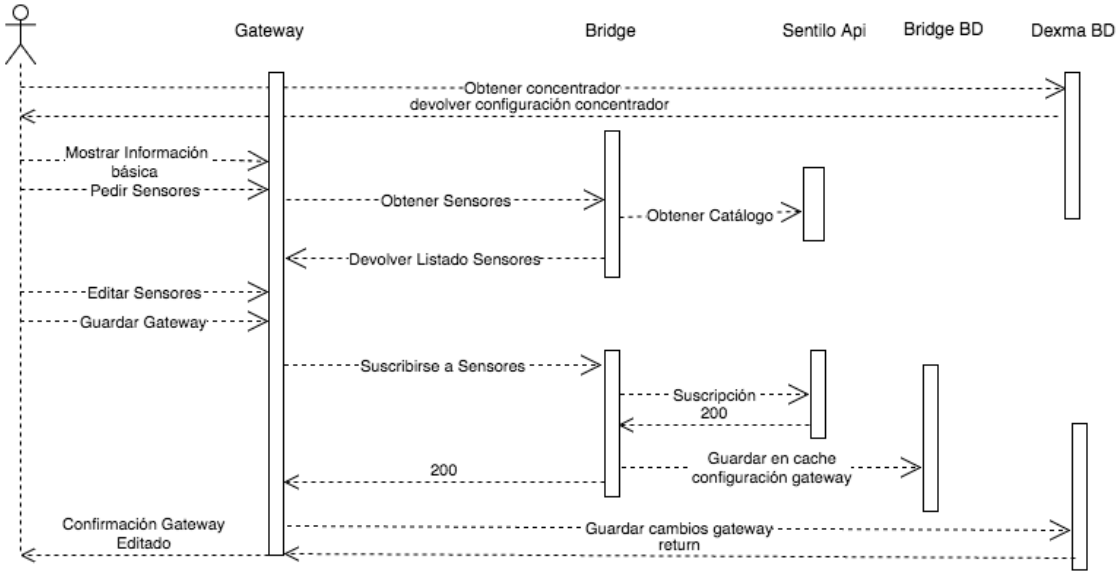


Ilustración 15 Diagrama comportamiento editar concentrador

Escenario Principal

1. El usuario selecciona el concentrador a editar
2. El sistema muestra a este la información del concentrador.
3. El usuario modifica la información básica de concentrador deseada menos el nombre
4. El usuario edita, elimina los dispositivos
5. El usuario guarda la información
6. El sistema actualiza la información del concentrador.

Escenario alternativo

5.a El usuario decide cancelar la edición del concentrador y conservar el estado anterior asi que aprieta cancelar

5.b El sistema muestra a este el listado de concentradores.

5.2.14 Editar dispositivos

Actor Usuario

Precondiciones El usuario debe haber creado previamente el concentrador en Dexcell con sensores asociados

Disparador El usuario se encuentra en el caso editar concentrador

Diagrama de comportamiento

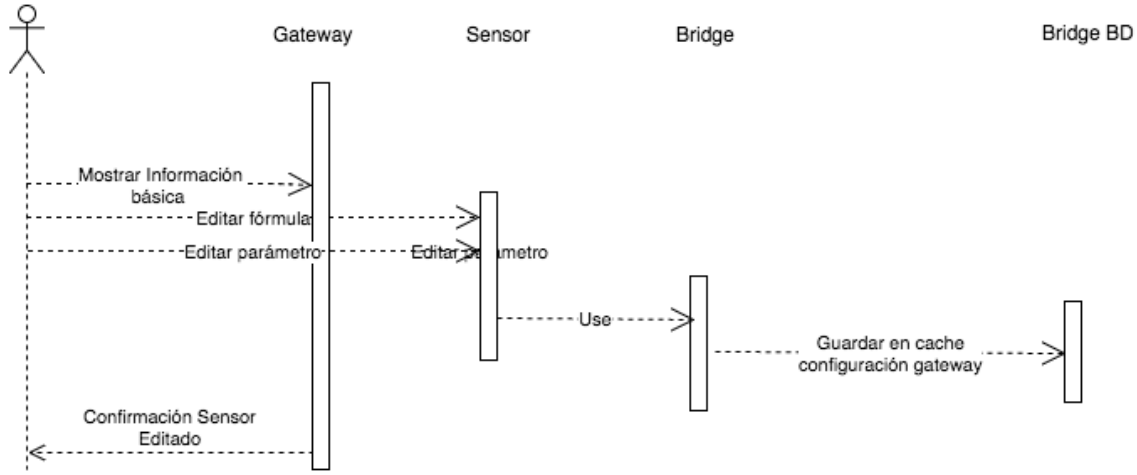


Ilustración 16 Diagrama comportamiento editar dispositivo

Escenario Principal

1. El usuario modifica la fórmula o parámetro asociados al sensor
2. El sistema guarda los cambios.

Escenario alternativo

2.a El usuario decide cancelar la edición del concentrador y conservar el estado anterior asi que aprieta cancelar

2.b El sistema muestra a este el listado de concentradores.

5.2.15 Servicio de importación de datos

Actor Sentilo

Precondiciones El bridge se ha suscrito al servicio de recepción de datos de Sentilo

Disparador -

Diagrama de comportamiento

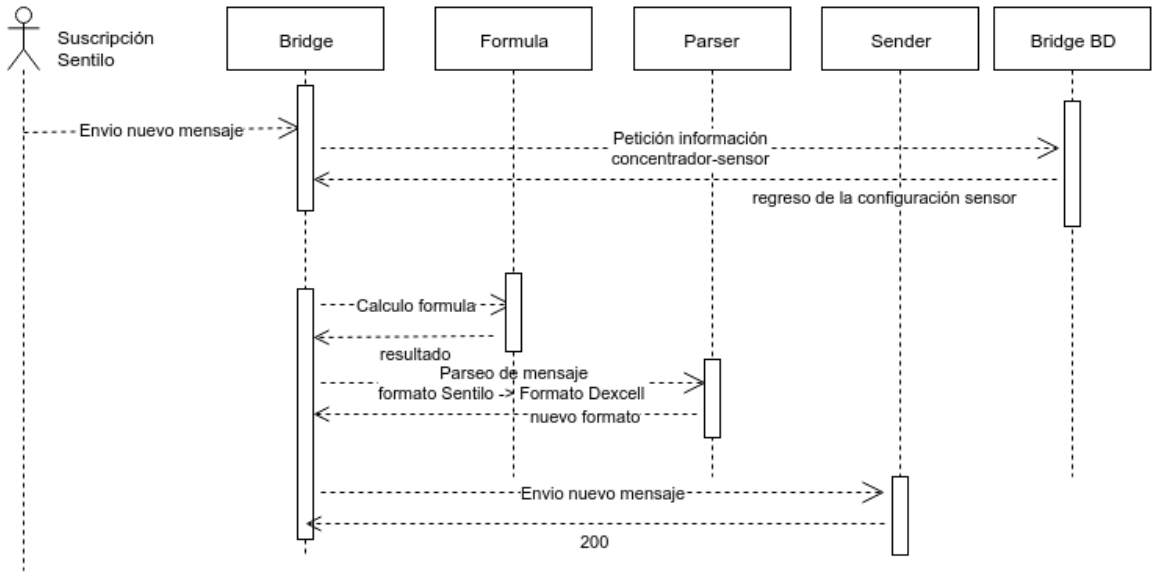


Ilustración 17 Diagrama comportamiento servicio importación de datos

Escenario Principal

1. Sentilo envía un mensaje al endpoint de nuestro sistema.

2. El sistema desgloça la llamada para obtener el concentrador suscrito solicitado y de su data obtiene el mensaje de Sentilo el cual consta de una data, un valor de consumo, el sensor del que proviene, y su proveedor, aunque nuestro sistema solo hará caso al valor de consumo y al sensor.

3. Una vez recogida toda la información de Sentilo el sistema solicitará la información del sensor usando como clave la unión de concentrador y sensor.

4. A continuación el sistema procederá a realizar la fórmula deseada por el usuario para transformar el valor recibido en el deseado en Dexcell.

5. El sistema seguidamente procederá a transformar en nuestro parser el formato del mensaje en uno entendible por Dexcell.

6. Para finalizar el sistema enviará al sistema de inserción de Dexcell el nuevo mensaje para que sea procesado y a su debido momento guardado en la base de datos de Dexcell.

Escenario alternativo

En caso de que alguna parte del proceso anterior al sender fallara, se procederá a guardar el mensaje de Sentilo en una cola de errores, en caso que falle el Sender se volverá a intentar un máximo de cinco veces, y en caso negativo se almacenará el mensaje de Sentilo en la cola de errores.

6 Diseño

El propósito de esta etapa es definir un sistema con suficiente detalle como para que sea posible su implementación, partiendo de la especificación previa.

A la hora de realizar el diseño de este proyecto se ha apostado por la modularidad de las diferentes capas y componentes para conseguir una cohesión alta a la vez que un acoplamiento bajo entre clases.

Para conseguir esto se ha optado por seguir un diseño DDD o “Domain-Driven Design” el cual se basa en dar relevancia a la capa de dominio aislándola de la capa de interfaz para de esta manera hacer más fácil su uso en distintas partes del proyecto y hacer más entendible la capa de negocio de nuestro proyecto.

Con el uso de este método DDD junto con el MVC o modelo-vista-controlador podemos ver desacoplados las distintas capas y se ha podido usar patrones típicos de DDD cómo serían las entities, repositories, services, el uso de estos patrones hará mucho más fácil la comprensión del proyecto para nuevos desarrolladores implicados en el proyecto y también facilita en gran medida su testeo y la independencia de las distintas clases gracias a la inyección de dependencias.

A continuación explicaremos de manera teórica algunos conceptos de diseño usados en este proyecto, como su arquitectura basada en DDD, principios de software usados o patrones usados.

6.1 Arquitectura (DDD)

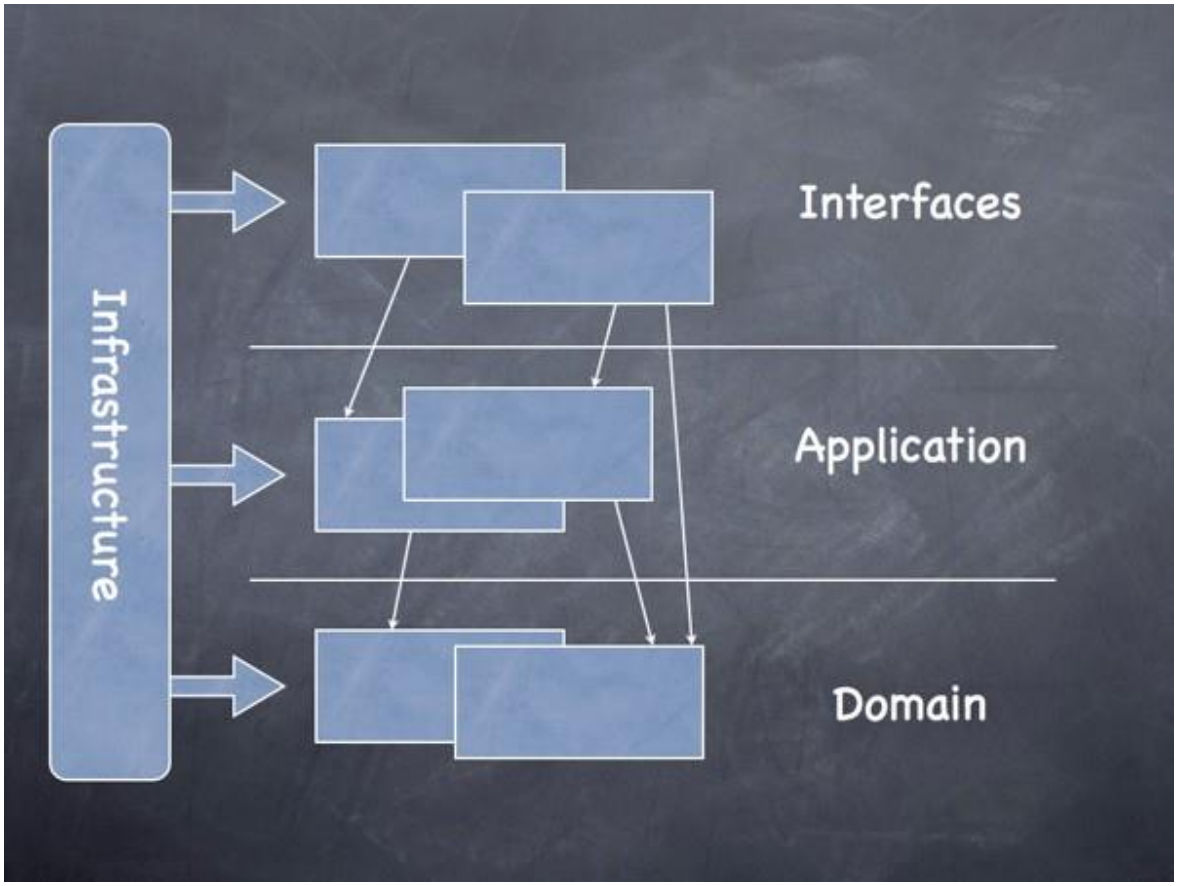


Ilustración 18 arquitectura Driven Domain Design

Como se puede ver en la ilustración 18 disponemos de tres capas: Interfaz, aplicación y dominio, cada cual soportados por diferentes tipos de infraestructuras

6.1.1 Interfaces

Esta capa contiene todo lo que interactúa con otros sistemas, como serian servicios web o aplicaciones web. Se ocupa de la interpretación, validación y traducción de los datos entrantes y de los que salen por esa interfaz. También es donde encontraremos las comprobaciones de seguridad propias de esa interfaz.

6.1.2 Aplicación

Esta capa es la encargada de dirigir el flujo de trabajo del aplicativo y la que suele coincidir con los casos de uso y son independientes de la interfaz que tengan por delante. Esta capa está muy pensada para el proceso de autenticación y la seguridad y la traducción del contenido de la capa de interfaz a modelos entendibles por el dominio.

6.1.3 Dominio

Esta capa es el corazón de un sistema basado en DDD y por tanto es donde la lógica interesante ocurre, en otras palabras, es el centro de la lógica de negocio, es el responsable de gestionar todas las entidades y servicios relacionados con el tema al que va dirigido la aplicación.

6.1.4 Infraestructura

En adición a las otras capas también encontramos la capa de infraestructura ocupada de dar soporte a las anteriores de varias maneras, facilitando la comunicación entre capas. En términos simples, la infraestructura consiste en todo lo que existe independientemente de nuestra aplicación: librerías externas, bases de datos, servidores web…

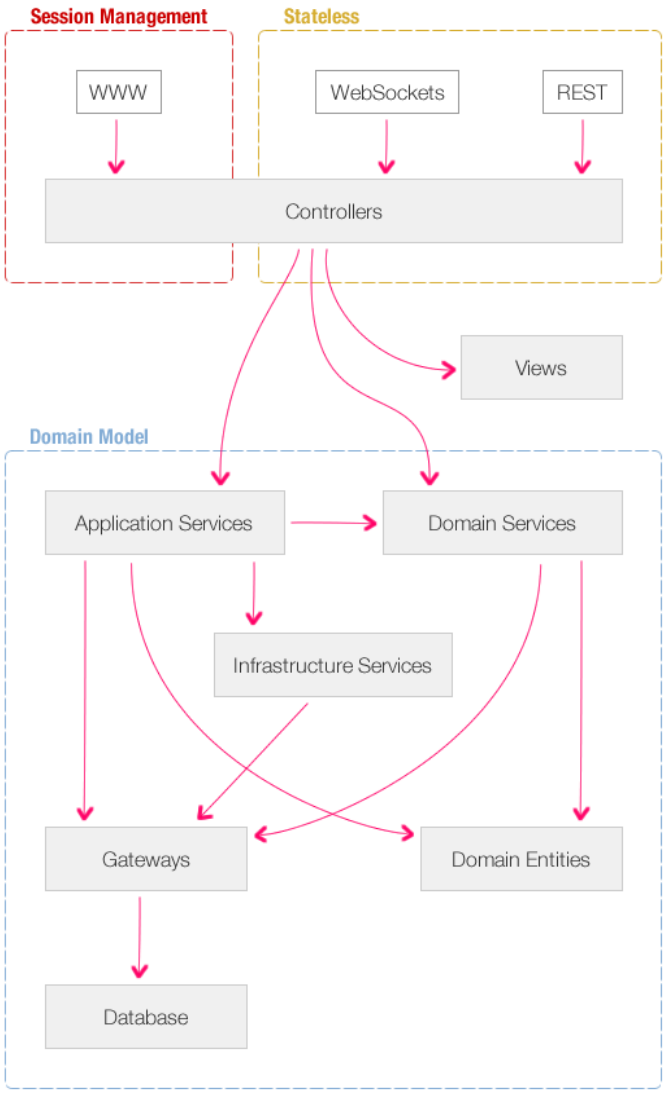


Ilustración 19 Ejemplo de infraestructura usando DDD

Viendo la ilustración anterior podemos ahora conectar las distintas capas para observar cómo sería el flujo entero de una aplicación:

- Un usuario quiere acceder a la funcionalidad de la aplicación y llama a esta, tanto si es mediante el aplicativo web, como si se trata de nuestro dispatcher
- La petición es recibida por la capa de aplicación que comprueba que todo sea correcto y hace las peticiones correspondientes al dominio.
- En este punto el dominio puede devolver directamente una entidad o realizar algún tipo de procesado en uno de sus servicios.
- Una vez obtenido el resultado el dominio devuelve este al nivel de aplicación que a su vez realiza lo propio con el de presentación.

6.2 Principios

6.2.1 “Don’t repeat yourself” (DRY)

El principio No te repitas (en inglés Don't Repeat Yourself o DRY, también conocido como Una vez y sólo una) es una filosofía de definición de procesos que promueve la reducción de la duplicación especialmente en computación. Según este principio toda pieza de información nunca debería ser duplicada debido a que la duplicación incrementa la dificultad en los cambios y evolución posterior, puede perjudicar la claridad y crear un espacio para posibles inconsistencias. Por "pieza de información" podemos entender, en un sentido amplio, desde datos almacenados en una base de datos pasando por el código fuente de un programa de software hasta llegar a información textual o documentación.

6.2.2 SOLID

Patrón que intenta asentarse sobre cinco principios que unidos deben ser capaces de crear un código más fácil de mantener y extender. Los cinco principios son

- **Principio de la responsabilidad única:** una clase debe poseer únicamente una responsabilidad
- **Principio abierto/cerrado:** las entidades deben ser abiertas para extensión pero no para modificación.
- **Principio Liskov substitution:** Objetos en un programa deben ser reemplazables con instancias de sus subtipos sin alterar la correctitud del programa.
- **Principio segregación de la interfaz:** Muchas interfaces cliente específicas son mejor que una interfaz de propósito general.
- **Principio de inversión de la dependencia:** Se debe depender de abstracciones no de implementaciones.

6.3 Patrones

A continuación vamos a hacer una breve mención de algunos de los patrones usados en este proyecto para luego poder hacer uso de sus nombres

6.3.1 Patron SmartUI (antipatrón)

Este fue el primer patrón usado en nuestro diseño, debido a que se tenía que sacar una primera versión del producto con urgencia se decidió realizarlo partiendo del llamado “anti patrón”, puesto que en un primer momento reduce el tiempo de desarrollo al tener para cada vista un único controlador que lo gestiona.

Este patrón no es recomendado puesto que promueve la repetición de código y dificulta el proceso de testeo del aplicativo. Por otro lado si que potencia la velocidad del desarrollo.

6.3.2 Patrón Repository

Patrón Encargado de mediar entre el dominio y la capa de almacenamiento de datos.
Los repositorios proveen de una fachada centralizada para enmascarar la tecnología desde donde recuperamos nuestros datos, tanto sea esta una API, una base de datos o simplemente un fichero.
En otras palabras: nos devuelven una entidad sin que el dominio se tenga que preocupar de donde ha venido.

6.3.3 Patrón Gateway

Este patrón sirve para crear un puente (o puerta de enlace) entre la capa de dominio dónde se almacena nuestra lógica de negocio con la capa de persistencia dónde se almacenan nuestros datos.
De esta manera si alguna vez decidimos cambiar la infraestructura de almacenamiento, sólo tendremos que volver a implementar estos “puentes”.

6.3.4 Patrón Factory

Un patrón de creación, destinado a la creación de objetos, tanto si estos son entidades como simplemente valores. La idea de usar este patrón es tener centralizado la creación de objetos, de esta manera para el usuario es más simple observar la estructura que debe seguir este objeto (cosa muy interesante en Python, puesto que al ser no definir los tipos de las variables, para objetos no primitivos suele ser muy difícil saber su estructura).

Este patrón por tanto ofrece desacoplamiento para la creación de objetos, una manera sencilla de crearlos y un módulo central robusto capaz no solo de crearlos sino también de analizar que estos cumplan las condiciones impuestas a ese objeto, tanto si se trata de los objetos que este debe contener como de los tipos de cada uno.

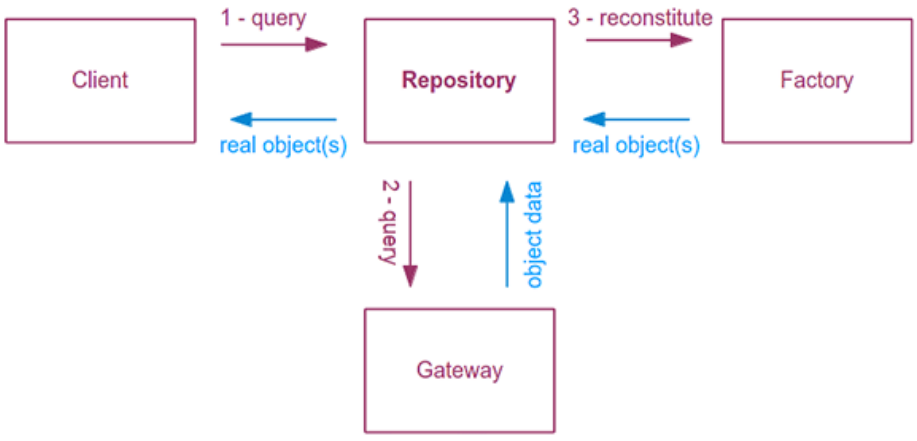


Ilustración 20 Diagrama Patrón Repositorio

En la figura anterior podemos observar el diseño resultante del patrón Repositorio, donde un cliente solicita una entidad a un repositorio y este pide a un Gateway de una tecnológica concreta que le devuelva la información sobre ese objeto que disponga para luego ser parseado en la factoria quien crea la entidad a devolver

6.3.5 Patrón Service

Parte del modelado del dominio, los servicios son usados en caso de que incorporar cierta funcionalidad como entidad pueda distorsionar el concepto de estas. Siguiendo la definición dada por Evans, un buen servicio cuenta de tres características:

- La operación hace referencia a un concepto del dominio que no forma parte naturalmente de una entidad o valor.
- La interfaz es definida en base a otros elementos del dominio.
- La operación no tiene ningún estado.

Los servicios sirven para expresar por tanto una serie de operaciones en la forma de un contrato, haciendo también de coordinadores.
Es donde tratamos las dependencias para poder crear, guardar o modificar una entidad, es decir, en caso de que por ejemplo para crear un tipo de entidad debamos antes comprobar si existen una cantidad de relaciones ya existentes.

A continuación vamos a mostrar cómo el diseño de nuestro proyecto por cada una de las diferentes partes, para que se pueda observar cómo hemos usado estas capas, patrones y principios, para eso empezaremos explicando la parte del proyecto dedicado a la exportación de datos, aplicación web, dispatcher y worker, y seguidamente explicaremos la importación de datos o en específico el bridge.

6.4Aplicación web

6.4.1 Capa de Vistas

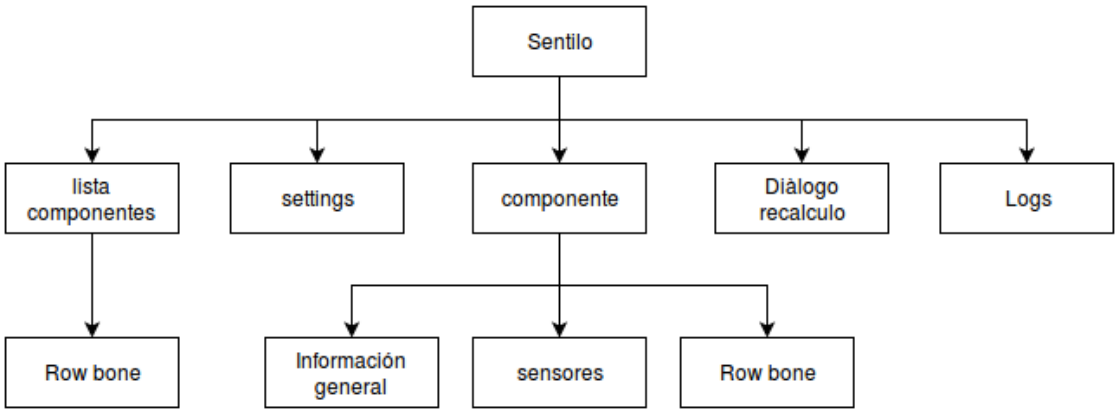


Ilustración 21 diagrama capa vistas aplicativo web

En la ilustración anterior podemos observar la cantidad de vistas usadas y como se relacionan entre ellas, como se puede ver la vista Sentilo en su interior contiene las demás. A continuación explicaremos con mas detalle las diferentes vistas:

Sentilo: la vista principal encargada de cargar todas las demás en su interior y mostrar la primera pantalla que es la lista de componentes.

Lista de componentes: En esta pantalla el usuario observará todos sus componentes en una tabla paginada (para no sobrecargar nunca nuestra base de datos).

Row bone (Lista de componentes): Esta vista contiene un template oculto que es la estructura de una fila del listado de componentes.

Settings: Esta vista contiene la configuración del usuario.

Componente: Esta vista contiene toda la información para crear un nuevo componente o actualizarlo.

Información General (Componente): aquí encontramos la información básica a rellenar para poder crear un componente vacío, es decir, sin sensores.

Sensores (Componente): Vista que contiene la estructura básica de la tabla para sensores.
Row Bone (Componente): Estructura básica de un elemento de la tabla de sensores.

Diálogo recálculo: Vista que encierra el diálogo encargado de recalcular cualquier componente.

Logs: Vista donde se encontrará el visor de los mensajes de estado de los componentes del usuario actual.

6.4.2 Capa de Interfaz

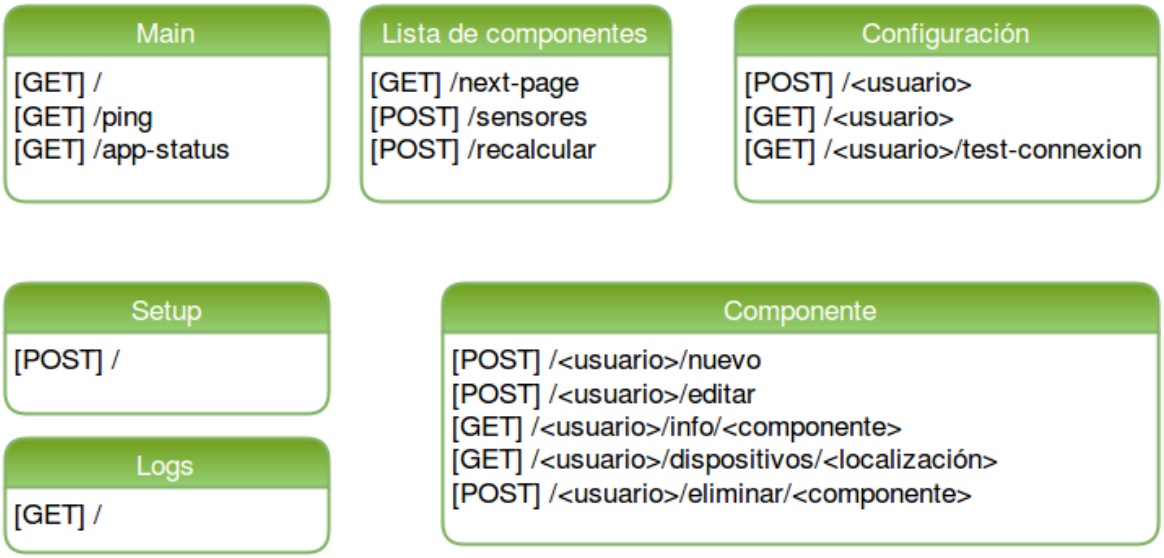


Ilustración 22 diagrama capa interfaz aplicativo web

En la figura anterior podemos observar las diferentes llamadas de las que dispone nuestro aplicativo permitiéndole conectarse con el mundo exterior, a continuación explicamos mas a fondo los distintos bloques.

Main: Este se ocupa de enviar al cliente toda la interfaz en una primera instancia y de los métodos ping y app-status usados para observar el estado de la aplicación.

- / llamada principal que devuelve todo el contenido html, javascript y css al cliente.
- /ping: llamada para monitorizar de manera simple desde sensu nuestra aplicación a través de un ping, simplemente devuelve un string pong.
- /app-status: llamada para monitorizar más a fondo nuestra aplicación devolviendo el estado de cada elemento de la aplicación, bases de datos, servidor web...

Lista de componentes: Esta interfaz contiene todas las llamadas destinadas a gestionar el listado de componentes.

- /next-page: Llamada encargada de obtener los siguientes veinte componentes para mostrar.
- /recalcular: llamada destinada a pedir un recálculo manual.
-

Configuración: Esta interfaz está destinada a gestionar la configuración de Sentilo.

- /<usuario>: la llamada post modifica la configuración, mientras que la get devuelve la configuración actual al cliente.
- /<usuario>/test-connexion: La llamada permite ver al usuario si su configuración es válida para conectarse con un servidor de Sentilo.

Setup: Esta interfaz es la primer usado por la aplicación, usada en el momento de instalación y se encarga de guardar la información necesaria para poder a posteriori conectarse a Dexcell a través de su API.

Logs: Esta interfaz se encarga de devolver al usuario los mensajes de estado de sus componentes guardados en nuestra base de datos.

Componente: Esta interfaz se encarga de contener todas las llamadas para gestionar la creación, modificación, visualización y eliminación de un componente y sus respectivos sensores.

- /nuevo: sirve para guardar un nuevo componente.
- /editar: sirve para actualizar algún campo de un componente ya existente.
- /info: devuelve la información de un componente al cliente.
- /dispositivos/<localización>: obtiene los dispositivos asignados a una localización de Dexcell.
- /eliminar/<componente>: Eliminamos de manera permanente el componente de nuestro sistema.

6.4.3 Capa de Aplicación

Esta capa usaría el diseño Smart UI puesto que para cada módulo de interfaz tenemos asociado un único módulo de aplicación y viceversa. Por tanto para cada módulo mostrado anteriormente, menos para el main, tenemos un controlador asociado con el mismo nombre.

Setup: Simplemente llama a los servicios de autentificación de Dexcell para realizar el proceso Oauth que devuelve el token para conectarse con la API de Dexma y a continuación delega en el servició de tokens para guardarlo en el sistema.

Lista de componentes: Se ocupa de mappear los datos recibidos del nivel de interfaz.

- Next page: se ocupa de pedir al dominio los siguientes componentes, devolviendo un listado de entidades de componentes al nivel de interfaz.
- Sensores: convierte el diccionario devuelto por la interfaz en entidades del tipo sensores y analiza que sean correctas antes de enviárselas al dominio para que las guarde en el sistema.
- Recalcular: Se ocupa de insertar en el servicio de colas la nueva petición para que sea procesada en su momento.

Configuración: En este caso la capa de aplicación no necesita orquestar nada, simplemente delegará todas las acciones a la capa de dominio, tanto la obtención de la configuración a través del servicio de usuario, como el salvaguardarla también a través de usuario. En el caso del testeo de la conexión también delegara en el servicio catálogo de Sentilo para observar si el proveedor especificado existe en el.

Logs: pedirá al servició de logs los últimos mensajes y los devolverá a la interfaz para que los formatee para la correcta visualización del usuario.

Componente: en este caso se ocupara de procesar la información de componente que le llega de la interfaz para convertirlo en entidades entendibles por el dominio. También se ocupara en el caso de eliminar un componente o editarlo de entender el mensaje devuelto por el dominio y convertirlo en un mensaje afirmativo o negativo entendible por la interfaz.

6.4.4 Capa de Infraestructura

En esta capa encontramos todas aquellas librerías y frameworks que nos han permitido el desarrollo de nuestra aplicación web de manera sencilla, para esta parte del proyecto cabe destacar el framework Flask mencionado en el apartado de tecnologías, también Celery y Babel framework dedicado a la traducción de frases, muy importante en nuestra aplicación ya que debe estar disponible en Inglés y Español.

6.5 Dispatcher

En este caso al ser un proceso lanzado por un cron y ser bastante simple no se ha dividido entre varias capas.

Dispatcher como se ha mencionado anteriormente, es el proceso encargado de obtener cada quince minutos los sensores configurados en nuestra aplicación web y delegar la tarea de exportar todos estos datos a los workers, en otras palabras dispatcher dispone simplemente de un nivel de aplicación que es levantado cada quince minutos por un cron del sistema que obtiene del servició de componentes de la capa dominio todos los componentes habilitados y cada uno de ellos lo encola en un sistema de colas esperando ser cogido por un worker que será el encargado de realizar la tarea de exportación.

6.6 Worker

En este caso nos encontramos con una capa interfaz parecida en diseño a la aplicación web, puesto que posee de un método que hace de receptor de las tareas que delega su funcionalidad a un módulo de aplicación ocupado de gestionar los diferentes servicios de dominio para obtener el componente, de este obtener los diversos sensores y con esto a través de servicios que delegan en ambas APIs obtener los datos de Dexcell del servicio de “readings”, parsear estos datos al formato de Sentilo y a través de un servicio data de Sentilo enviar los consumos a Sentilo.
Su otra responsabilidad es ir rellenando paralelamente a toda su ejecución nuestro servicio de logs para que el usuario pueda visualizar el estado actual de sus sensores.

6.7 Capa de Dominio

En esta capa encontramos los servicios, repositorios, gateways, factories y objetos compartidos por todo el proyecto. Aunque los únicos ocupados de comunicarse con las capas de encima son los servicios.

Servicio de colas: servició ocupado de configurar y lanzar nuestro sistema de colas para que tanto la aplicación web como el dispatcher puedan insertar

datos en ella y también para configurar los workers para que sepan en que cola leer.

Servicio de componente: encargado de devolver tanto una entidad especifica de componente, como actualizarlos, eliminarlos o crearlos, para todo ello este servicio delega en un Gateway especifico de la base de datos usado y de un Factory que mapea el resultado de este en una entidad del tipo componente entendible por todo el dominio. En el caso del componente que tanto se le accede desde nuestra base de datos como desde la API de Sentilo dispone de dos gateways distintos, uno destinado a conectarse con la base de datos propia y otro destinado a gestionar la comunicación con la API de Sentilo.

Servicio extracto componente: Este servicio esta pensado especialmente para devolver un listado de componentes pero solo devolviendo por cada uno la información básica de un componente, información mostrada en el listado de componentes, este servicio a sido creado principalmente para reducir la carga de información que debe devolver nuestra base de datos al devolver el listado de los diversos componentes.

Servicio Sensor: Este servicio esta principalmente dedicado a obtener los diversos sensores de un proveedor, suscribirse a un nuevo sensor. (para el importador) y a guardar nuevos sensores en Sentilo (el exportador).

Servicio de usuarios: ocupado de realizar el guardado, actualización y obtención de la entidad usuario de la base datos que contiene básicamente la configuración de Sentilo usado, al igual que los servicios anteriores delega en un Gateway y Factory propios.

Servicio de tokens: ocupado de realizar el guardado y obtención de tokens de la base datos para cada usuario, al igual que los servicios anteriores delega en un Gateway y Factory propios.

El resto de servicios ocupados de interactuar con la API de Dexcell han sido implementados por separado puesto que se ha creído importante realizar para ellos una librería que no solo sea usable en este proyecto sino que pueda ser reusado en cualquier otro proyecto en Python que use la nueva API de nuestra empresa, tanto si es un proyecto interno como uno externo.

6.8 Dexcell APIv3

Esta es la capa o librería encargada de devolver los servicios relacionados con la nueva API de dexcell, para simplificarla se ha realizado una capa de diseño muy similar a la API de google cloud que a través de una clase build construye cualquier servicio disponible en la API, es decir, puede construir un servicio ocupado para la gestión de localizaciones, de dispositivos, de lecturas, de costes, de suministros…

6.9 Bridge (Importador)

Este es el proceso dedicado a recibir los datos de Sentilo y insertarlos en el sistema de inserción de Dexcell, para ello dispone de una capa de interfaz en modo API que permite recibir las diferentes peticiones tanto de Sentilo como de Dexcell, una capa aplicación dedicada a formatear esta información para el dominio y el dominio ya comentado anteriormente.

6.9.1 Capa de interfaz

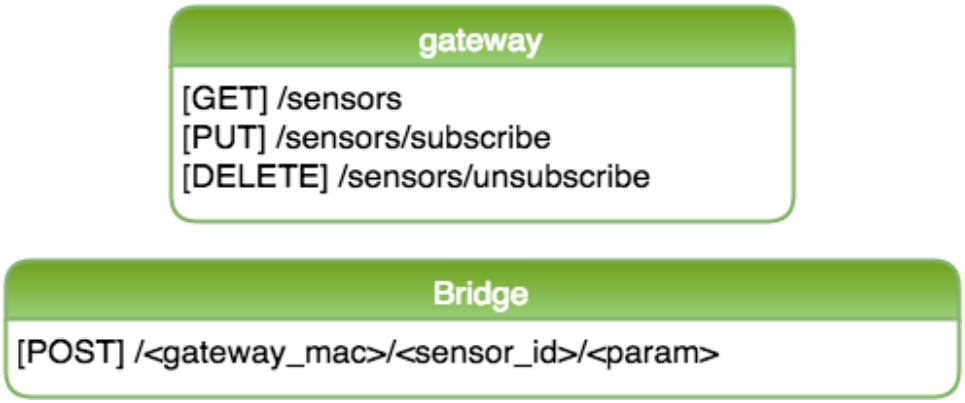


Ilustración 23 capa interfaz Bridge

Como podemos ver en la ilustración 23 tenemos dos partes, gateway o concentrador esta dedicado a ser usado como API interna para el concentrador creado en Dexma para realizar todas las interacciones necesarias con Sentilo

- **Sensors (GET) :** Esta llamada devolverá el listado de sensores para un proveedor para una dirección URL y token pasados desde Dexcell.
- **Sensors suscribe (PUT)** : Esta llamada realizará la suscripción a Sentilo de todos los sensores pasados desde Dexcell, para la suscripción se deberá pasar una dirección de recepción de datos (bridge).
- **Sensors unsuscribe (DELETE)** : Esta llamada recibe las peticiones de Dexcell para la cancelación de la suscripción de un listado de sensores y realiza esta petición a Sentilo.

Para el bridge la única llamada necesaria era la usada por Sentilo para traer los datos a Dexcell, esta llamada contiene toda la información necesaria para que se pueda transformar el valor al deseado por el dispositivo de Dexcell a través de varios procesos.

6.9.2 Capa aplicación

En el caso de Gateway simplemente transmite los datos al servicio de suscripción o catálogo del dominio, en el caso del bridge realiza la coordinación de las tres partes necesarias que son, el cálculo del valor a insertar a Dexcell a través de una fórmula, la conversión de formato de mensaje hacia el solicitado por Dexcell y la inserción a través de la API.

7 Arquitectura de la plataforma

7.1 Introducción

Para el proyecto de exportación y el bridge usado en la importación se ha decidido el uso de Python como lenguaje nativo del servidor, con Flask como framework web, mientras que para la configuración del importador se ha usado Java.

Para las bases de datos se ha decidido usar Redis y MongoDB.
Para el sistema de colas distribuidas se ha optado por celery junto con RabbitMQ.

Para el testeo del funcionamiento del aplicativo por parte del servidor tanto funcional como unitario se ha optado por pytest.
Por último para el desarrollo del lado de cliente se ha optado por html, css3 y javascript (ecma script 5) con el uso de librerías típicas como Jquery.

El aplicativo está alojado actualmente en google cloud.

Se decidió usar python en vez de otros lenguajes como java, PHP o ruby puesto que a nivel de codificación es mucho más simple y entendible que java o PHP ya que obliga a seguir unas pautas sintácticas bien marcadas y encima por lo general reduce mucho las líneas de codificación, por ejemplo, un código de 1000 líneas de java se puede ver resumido en 200 en un código python. La decisión entre Ruby y Python se tomó básicamente a favor de Python puesto que es uno de los lenguajes más usados dentro de la empresa, mientras que Ruby por el momento solo ha sido usado para testear junto con el framework Cucumber.

Para la configuración del concentrador si que se ha usado java como lenguaje nativo del servidor, puesto que el aplicativo de Dexcell fue realizado en este lenguaje.

La decisión de las bases de datos fue más rigurosa. La opción de Redis era básica para guardar un conjunto de información indispensable de la aplicación que debía estar disponible y accesible rápidamente en cada ejecución de cualquier usuario. Por otro lado la de Mongo fue la más pensada, por un lado hacía falta una base de datos que nos permitiera guardar datos de tamaño fijo como es el caso de la configuración propia de cada usuario, pero también que permitiera duplicar ciertos campos. Por otro lado la velocidad de lectura en este caso fue considerada más relevante que la de escritura, puesto que cada hora se deben leer todos los documentos, por todas estas razones se eligió MongoDB puesto que permite rápidas lecturas y fáciles modificaciones de sus documentos para añadir nuevos campos.

El testeo, parte ya fundamental de cualquier aplicación moderna que se precie, fue pensado ser realizado con un framework llamado Nose, pero al final se apostó por pytest que a pesar de ser muy parecido aporta funcionalidades complementarias muy interesantes como el poder insertar tags en los tests. El testeo se comentará con mas detalles en el siguiente capítulo que esta totalmente dedicado a explicar como se ha testeado el proyecto.

La automatización del sistema se realizó con Jenkins, que a pesar de ser el más pobre en funcionalidades es open source y por tanto 100% económico, aparte que consta con las funcionalidades necesarias para el despliegue automatizado de una aplicación como la nuestra.

Para desplegar el sistema en un primer momento se optó por heroku por su facilidad y su bajo coste, a pesar de todo al poco tiempo se migró a google cloud siendo este mas robusto, mas potente y con más opciones tanto de seguridad como para distribuir la aplicación entre diversos servidores en diversas zonas horarias, aparte de ofrecer opciones para incorporar una DMZ y una máquina bastión dedicada a ser la única puerta de entrada a la DMZ.

Para el entorno de desarrollo se decidió usar Vagrant para crear entornos limpios que emulan el entorno de producción y que permiten poder crearlos y

destruirlos en cuestión de segundos facilitando así la faena del desarrollador, este sistema también ha sido muy útil para la etapa de integración y testeo.

Para la monitorización del sistema se empezó usando Nagios aunque hace poco se pasó a usar Sensu un sistema de monitorización de sistemas y aplicaciones más potente y flexible destinado sobretodo a servicios web.

A continuación procederemos a explicar mas en detalle las tecnologías usadas, exponiendo las ventajas que suponen.

7.2 Frameworks Webs

7.2.1 Flask

Flask es un micro framework web para Python creado por Armin Ronacher, basado en Werkzeug y Jinja2 con licencia BSD, por tanto de código abierto. El framework fue publicado por primera vez el 1 de abril de 2010 y llegó a su versión estable tres años después en el 14 de junio de 2013.

El hecho que se base en Wekzeug implica que es un framework 100% basado en WSGI (“web server gateway interface”) una interfaz simple e universal a bajo nivel para conectar aplicaciones webs con el lenguaje Python.
WSGI fue originalmente definido en el PEP 333 que se convirtió en el estándar para desarrollar aplicaciones web en Python.

Por otro lado el uso de Jinja 2 supone más que un acierto para la comunidad de Flask puesto que actualmente es considerado el lenguaje de “templates” más moderno, rápido, seguro y amigable para Python.

La principal ventaja de Flask respecto a otros micro frameworks webs es que se basa en la filosofia “small core and easy-to-extend”, es decir, a pesar que en su base es tan limitado como, por ejemplo Bottle, tiene una de las

comunidades pythoneras más extensas y está en constante desarrollo lo que ha posibilitado la creación de multitud de extensiones que permiten llevar a Flask a un nivel de complejidad y usabilidad tan grande como Django.

Otra ventaja de Flask es que al ser un framework tan joven y posterior a otros de renombre como Django o bottle ha tenido tiempo de aprender de sus errores y ha tener un carácter más “Pythonic”, puesto que se basa en una filosofía básica de python, el DRY o “don’t repeat yourself”, que permite levantar una aplicación web de manera mucho más simple ya que elimina la repetición de código y complejidades innecesarias para pequeños aplicativos.

A pesar de que Flask en un origen no está basado en el patrón MVC o modelo vista controlador gracias a todas sus extensiones es fácilmente integrable.

7.3 Bases de datos

7.3.1 Redis

Una base de datos no relacional basado en clave-valor con licencia BSD (es decir también código libre) creado por Salvatore San Filippo en lenguaje C cuyo primer lanzamiento fue el 10 de Abril de 2009 y cuyo nombre significa “REmote DIctionary Server”.

El hecho que sea una base de datos clave-valor implica que está orientado a guardar, devolver y manejar arrays asociativos o comúnmente diccionarios o hash, es decir, una estructura de datos en que cada valor corresponde a una clave única que lo identifica en todo el diccionario. Esto implica que cada valor guardado en Redis puede tener un formato distinto o incluso ser de un tipo distinto facilitando un mejor uso de la memoria. Redis por tanto usa el paradigma OOP “object oriented programming”.

A continuación expondremos el papel de Redis en algunas propiedades básicas de las bases de datos:

- **Persistencia**: Como hemos mencionado anteriormente por defecto Redis guarda toda la información en memoria, es decir, en una unidad volátil, a pesar de todo Redis consta de dos opciones para la persistencia de la información.
    - Snapshotting: es una persistencia a medias, puesto que lo que se realiza es una copia de los datos de memoria a disco de manera asíncrona, por tanto cabe la posibilidad de perder los datos desde el último snapshot.
    - AOF: Se guardan todas las operaciones de escritura, de manera que en caso de caída, al levantarse de nuevo Redis ejecutará de nuevo todas estas operaciones guardadas para recuperar el estado actual.
- **Replicación**: Gracias al uso de Redis Sentinel, Redis soporta el paradigma amo-esclavo, donde un esclavo puede ser amo de otros esclavos formando asi una estructura en arbol.
- **Rendimiento**: Al ser una base de datos que corre en memoria, su velocidad de escritura y lectura es infinitamente mayor que la del resto de bases de datos que deben interactuar con el disco.

7.3.2 MongoDB

Una base de datos no relacional orientada a documentos diseñada en MongoDB Inc, escrito en C/C++ y javascript bajo licencia BSD y cuya primera versión se publicó en 2009.

Está basado en documentos que es una subclase de clave-valor, aunque esta difiere en que los documentos se suponen estructurados con una metadata común y que la base de datos usa para optimizar búsquedas.

Una de las principales diferencias de MongoDB con una base de datos relacional, es que Mongo se basa en desnormalizar las tablas relacionales para juntarlas todas en un solo documento y asi hacer la extracción de la información más rápida.

A continuación expondremos el estado de Mongo en algunas propiedades básicas de las bases de datos:

- **Indexación**: Mongo permite crear índices para todos sus documentos, y si estos caben situarlos en memoria. En caso de índices muy grandes (cosa que implicaría un fallo en el diseño) se localizan en el disco.
- **Replicación**: usa un modelo donde encontramos una instancia que hace de primario, es decir, al que todas las escrituras se dirigen, y todas las otras instancias son llamados secundarios que se ocupan de las lecturas. La elección del primario es realizado a través de votaciones y por tanto es recomendable siempre tener un número impar de instancias que puedan votar.
- **Load balancing**: Mongo permite escalar en horizontal a traves de sharding, es decir se escoge una clave, que será la que separa los datos en diversos shards y dentro de estos usaran la metodología master slave.

7.4 Sistemas de colas

7.4.1 Celery

Es un encolador de tascas asíncronos basado en distribuir mensajes en tiempo real, que también permite crear trabajos programados.

Se basa en tener un proceso cliente que envía las tareas a un broker, el cual puede ser entre otros un redis o rabbitMQ y que se ocupa de situarlas en una cola a la espera que uno de sus trabajadores pida la siguiente tarea. Este proceso permite derivar las tareas en tantos trabajadores como creamos convenientes, puesto que permite escalabilidad horizontal entre diversos servidores y en un mismo servidor el uso de tantos trabajadores como unidades de proceso tengamos.

7.4.2 RabbitMQ

Desarrollado como software libre bajo la licencia de Mozilla Public License, se trata de un transportista de mensajes o “broker” que implementa AMQP o “advanced message queuing protocol” sobre el lenguaje funcional Erlang.

El proyecto RabbitMQ consta de diferentes partes:

- El servidor de intercambio RabbitMQ en sí mismo
- Pasarelas para los protocolos HTTP, XMPP y STOMP.
- El plugin Shovel (pala) que se encarga de copiar (replicar) mensajes desde un corredor de mensajes a otros

7.5 Frameworks testing

7.5.1 Pytest

Como indica el nombre, este es un framework diseñado para el testeo. Desde los simples tests unitarios hasta los más complejos tests funcionales.

Sus ventajas respecto a otros frameworks de testeo de python son:

- Provee testeo siguiendo la filosofia DRY o “don’t repeat yourself”.
- Ofrece la cantidad de extensiones más completa de entre los frameworks de python.
- Permite paralelizar los tests entre las diversas CPU’s incrementando asi la velocidad de resolución.
- Permite la inclusión de tags o otros elementos de referencia para poder ejecutar solamente un subconjunto de todos los tests.
- Está perfectamente integrado con otros frameworks como Nose, Unitest o Doctests.

7.6 Entorno Desarrollo

7.6.1 Vagrant

Como se define en su página web, Vagrant crea y configura entornos de desarrollo ligeros, reproducibles y portátiles.

Algunas de las ventajas de Vagrant son:

- Permite Carpetas compartidas entre el cliente y el entorno creado
- Permite levantar varios entornos virtuales para simular entornos distribuidos, es decir, por ejemplo podemos tener tres instancias, una con un servidor web, otro con un Mongo y otro con un postgres.
- Permite diversos modelos de configuración de red.
    - Forwarded host: como ventaja tiene que hace visible a la maquina virtual desde fuera del host asignando puertos del host a la maquina, en su contra tiene su lentitud y la imposibilidad de conectar varias VM entre ellas.
    - Host only networking: Se crea una red privada entre el host y la VM a través de una IP estática, este método aumenta la seguridad puesto que solamente permite el acceso al host y a otras VM instaladas en el host y encima permite comunicación bidireccional de datos entre host y VM.
    - Bridget Networking: puente que une la VM a un dispositivo, su funcionamiento es parecido al “host only networking”, aunque este es visible desde toda la red interna, pero no dispone de IP estática sino que cada vez se le asigna una nueva.
- Permite creación de “cajas” propias, es decir, snapshots de sistemas operativos diseñados a la medida de las necesidades de cada proyecto, para entenderlo mejor, podríamos crear una caja de Ubuntu donde dentro ya tuviera cargado java , de manera que cada vez que la cargaramos no tuviéramos que perder un buen rato esperando a que se instale.
- Permite integrarse con herramientas de automatización como Ansible, Puppet o Chef.

7.7 Herramientas de Integración continua

7.7.1 Jenkins

Basado en el proyecto Hudson, Jenkins es uno de los pocos sistemas de integración continua open source que existen actualmente.

La base de Jenkins son las tareas, donde indicamos qué es lo que hay que hacer en un build. Por ejemplo, podríamos programar una tarea en la que se compruebe el repositorio de control de versiones cada cierto tiempo, y cuando un desarrollador quiera subir su código al control de versiones, este se compile y se ejecuten las pruebas. Si el resultado no es el esperado o hay algún error, Jenkins notificará al desarrollador, al equipo de QA, por email o cualquier otro medio, para que lo solucione. Si el build es correcto, podremos indicar a Jenkins que intente integrar el código y subirlo al repositorio de control de versiones.

Pero la cosa no queda ahí! Una de las cosas buenas que tiene Jenkins es que además de poder ayudarte a integrar el código periódicamente, puede actuar como herramienta que sirva de enlace en todo el proceso de desarrollo.

Desde Jenkins podrás indicar que se lancen métricas de calidad y visualizar los resultados dentro de la misma herramienta. También podrás ver el resultado de los tests, generar y visualizar la documentación del proyecto o incluso pasar una versión estable del software al entorno de QA para ser probado, a pre-producción o producción.

7.8 Herramientas de automatización

7.8.1 Ansible

Software libre creado en el año 2012 que permite configurar y manejar sistemas operativos de una manera simple, limpia y eficaz. Adicionalmente Ansible es categorizada como una herramienta de orquestación.

Ansible usa playbooks basados en YAML para definir las acciones a realizar.

Como la mayoría del software para administrar configuraciones, Ansible distingue dos tipos de servidores: controladores y nodos. Primero, está una única máquina de control donde la orquestación comienza. Los nodos son manejados desde esa máquina de control por SSH. La máquina de control conoce a los nodos a través de un inventario.

Para organizar los nodos, Ansible despliega módulos a los nodos via SSH. Los módulos son guardados temporalmente en los nodos y se comunican con la máquina de control a través del protocolo JSON sobre una salida estándar. Cuando Ansible no controla los módulos, estos no consumen recursos porque no existen procesos o programas ejecutándose en segundo plano.

En contraste con otros programas de control como Chef o Puppet, Ansible usa una arquitectura sin agentes, es decir, los nodos no necesitan instalar ni ejecutar en segundo plano ningún proceso que se comunique con la máquina de control. Este tipo de arquitectura reduce la sobrecarga de la red y previene el uso de estrategias de control más agresivas por parte del servidor.

El diseño de Ansible incluye:

- Mínimo por naturaleza, es decir, los sistemas de administración deben disponer de lo necesario y nada más.
- És consistente.

- Seguro, al usar solo openSSH que es considerado crítico y altamente testeado, Ansible puede asegurar que no habrá vulnerabilidades al instalar en nodos.
- Alta confiabilidad. El modelo de idempotencia es aplicado para las instalaciones y configuraciones, para prevenir efectos secundarios en la ejecución repetitiva de scripts.

7.9 Herramientas de monitorización

7.9.1 Sensu

Sensu es un software libre escrito en Ruby nacido por la necesidad de una nueva herramienta de monitorización adaptada a sistemas en la nube, es decir Sensu incorpora los siguientes elementos mancantes en Nagios:

- Es adaptable al cloud, deja de lado la rigidez y un sistema estático.
- Aporta nuevos modelos de configuración más adaptables y entendibles usando el formato JSON.
- Es fácil extender.
- Permite descubrir nuevos servidores de manera automática.

Para conseguir estos puntos Sensu recurre a un sistema de colas, RabbitMQ que le permite encolar de manera segura peticiones y resultados de cualquier servidor haciendo posible el escalado del sistema sin necesidad de interrupciones, puesto que cualquier nueva conexión simplemente es encolado en RabbitMQ.

Otro punto fuerte de Sensu es el uso de Redis para el almacenaje de los datos de eventos o clientes.

7.9.2 Rabbitmq-management plugin

Herramienta de monitorización exclusiva para el sistema de colas RabbitMQ, básicamente provee de una API por HTTP que permite el manejo y monitorización de tu servidor RabbitMQ

7.10 Servidores Web

7.10.1 UWSGI

UWSGI es un proveedor de servicios de hosting para aplicaciones Python, para esto dispone de una amplia API que proporciona un servidor para aplicaciones para distintos lenguajes y distinto protocolos.

La gran ventaja de UWSGI es que a pesar de proporcionar otros protocolos, está sobretodo centrado en el estándar de Python WSGI y supone el servicio de hosting para Python más optimizado en todos los sentidos, desde el más versátil, al que proporciona mayor rendimiento y con el menor uso de recursos del sistema (memoria, CPU, disco...)

Algunas otras ventajas de UWSGI serían las siguientes:

- Está conectado con libpython, esto permite que cargue el código de las aplicaciones al arrancarse y que actúe como un intérprete de Python.
- Viene con soporte directo para NGINX.
- Está escrito en C.
- Desde el 2013 está en una fase de continuo desarrollo y mejoramiento.

Gracias al plugin Emperor, UWSGI se dota de toda una serie de propiedades de monitorización y administración de sus aplicaciones y de sí misma.

7.10.2 Nginx

Nginx es un servidor web open source, focalizado en permitir un nivel alto de concurrencia, rendimiento y un uso reducido de la memoria. También puede ser usado como proxy inverso o servidor para correo electrónico.

Nginx está totalmente preparada para integrar aplicaciones que usan WSGI, por tanto esto la convierte en ideal para integrar con aplicaciones hechas en Python.

Una gran ventaja de Nginx es que al poder usarse como proxy inverso permite distribuir la aplicación entre diversos servidores de manera sencilla y asi distribuir la carga de conexiones a este.

```
Client          Request          ---->          Nginx          (Reverse-Proxy)
                 |
                /|\
               |      |      `->      App.      Server      I.      127.0.0.1:8081
               |       `-->       App.       Server       II.       127.0.0.1:8082
                `----> App. Server III. 127.0.0.1:8083
```

7.11 Lado Cliente

7.11.1 Materialize Design

Creado y diseñado por Google, Material Design es un lenguaje de diseño que combina los principios clásicos del diseño exitoso junto con la innovación y la tecnología. El objetivo de Google consiste en desarrollar un sistema de diseño que permite una experiencia de usuario unificada a través de todos sus productos en cualquier plataforma.

La metáfora de material define la relación entre el espacio y el movimiento. La idea es que la tecnología es inspirada en el papel y la tinta y se utiliza para facilitar la creatividad y la innovación. Las superficies y los bordes proporcionan indicaciones visuales familiares que permiten a los usuarios comprender rápidamente la tecnología más allá del mundo físico.

7.11.2 Jquery

Es una librería de Javascript ligera con un solo propósito, hacer a los desarrolladores mucho más sencillo el uso de Javascript en el navegador, simplificando las funciones y haciéndolas mas entendibles.

7.12 Dexma

7.12.1 API interna

Para las conexiones contra DEXCell energy manager usamos la API de la que dispone, esta fue reformada hace pocos meses con una estructura más robusta y tratando de usar una serie de convenciones que se creyeron apropiadas para hacerla más entendible, lógica y sobretodo dar cierta coherencia estructural entre las diversas llamadas. Entre las diversas convenciones podemos encontrar algunas como:

- Las nomenclatura de las acciones que representan una petición a través del protocolo HTTP (get, post, put …) debe de estar compuesta por verbos y no por nombres.

- Todos los nombres siempre estarán en plural, puesto que todas las llamadas permiten la devolución de más de un objeto.
- Uso del principio KISS, es decir, mantener los nombres de las url simples.
- Todas las respuestas de las peticiones http tienen que estar formadas cómo un objeto JSON, debido a que es uno de los formatos de datos más usado actualmente.
- Las respuestas que contengan una gran cantidad de información deberán de dividirse en varias. Para poder acceder a cada una de ellas utilizaremos un sistema de paginación (con enlaces a las demás respuestas).

Un tema muy importante en el uso de la API de Dexma es la seguridad, y para este se han realizado dos etapas, la primera para la autorización y la segunda para la autentificación.

Autorización: la API hace uso de tokens para analizar si una llamada tiene permiso para entrar en el sistema, en concreto cada llamada realizada debe añadir una cabecera con la llave **x-dexcell-token** que identifica al usuario dentro del sistema.

Autentificación: esta se encarga de permitir o denegar acceso a un recurso concreto observando una serie de normas que determinan qué usuarios tienen acceso a qué recursos, estos permisos están directamente relacionados a cada aplicación. Las autorizaciones están divididas en dos conceptos:

- **Alcance:** determina si para un recurso concreto una aplicación tiene permisos de escritura/lectura. Este alcance es definido en la creación de la aplicación dentro de DEXcell.
- **ACL (lista de control de acceso):** Este es el último paso antes de su acceso al sistema, cada recurso dentro de DEXcell posee un identificador que permite ver si un usuario tiene acceso al recurso.

Una cosa a tener en cuenta con la API de DEXcell es que tiene un límite de llamadas por token tanto por día, como por hora. Este factor permite una distribución más equitativa de los recursos entre los diferentes usuarios y también limita sobrecargas en el sistema.

Ilustración 24 Diagrama pasos seguidos por la API

En la ilustración anterior podemos ver explicados los diferentes pasos que sigue una llamada antes de obtener la auténtica entidad a devolver, primero están las diferentes autentificaciones y el análisis que comprueba si

disponemos aun de llamadas para ese día y hora y en última instancia si todo a salido correctamente nuestra devolución del recurso con un 200!

También podemos observar como en caso de que nuestra llamada no pase el proceso de autorización la API de dexcell nos devolverá el error correspondiente.

8 Testeo y juego de pruebas

Este apartado como bien indica el título está orientado a explicar los procedimientos de testeo usados en este proyecto.

Lo primero que se debe entender es que actualmente no hay ningún software que se precie que no conste de su correspondiente testeo, puesto que este procedimiento aparte de permitirnos comprobar posibles fallos, nos permite comprobar la potencia de carga que puede soportar nuestro código, y en muchos casos sirve como punto de referencia para nuevos desarrolladores para entender el código actual .

Por todas las razones anteriores y muchas más he considerado importante dedicar unas páginas a explicar cómo y con qué herramientas he realizado estos procesos en el proyecto.

Todo el testeo propio de funcionamiento del código se ha realizado con una librería llamada pytest, la cual permite una gran cantidad de funcionalidades entre las cuales parte de las habituales en cualquier librería de testing podemos encontrar otras tan útiles como incorporar de tags a los diversos tests, para en el momento de ejecución solo lanzar los que nos interesen en ese momento.

A continuación vamos a ir comentando las diferentes etapas de testeo desde el testeo unitario, realizado a la vez que la implementación de los bloques, hasta llegar a los tests de volumen y estrés.

8.1 Prueba unitaria

8.1.1 Explicación

En programación, una prueba unitaria es una forma de comprobar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. Luego, con las Pruebas de Integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión.

La idea es escribir casos de prueba para cada función no trivial o método en el módulo, de forma que cada caso sea independiente del resto.

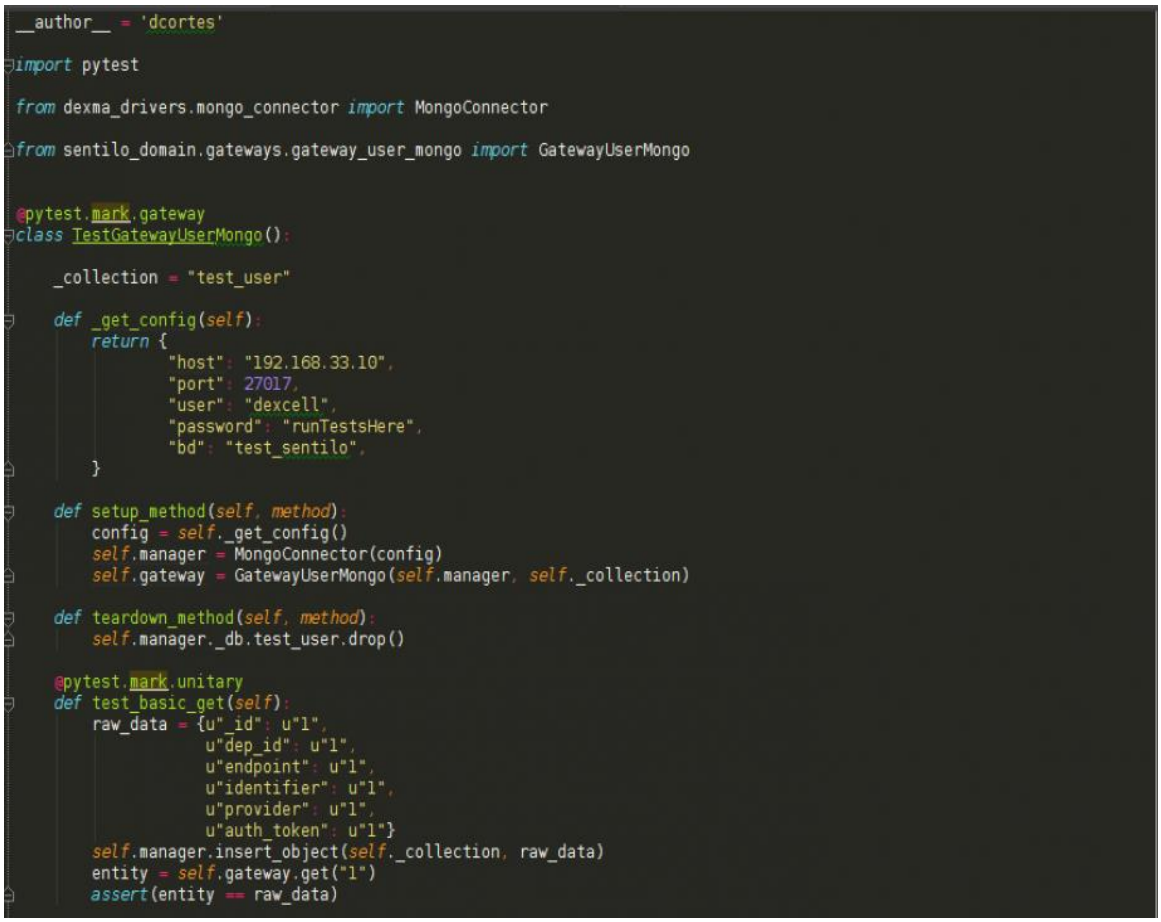
Para hacer de este testeo unas pruebas de calidad se recomienda siempre que cumplan las siguientes características:

- **Automatizable:** No debería requerirse una intervención manual.
- **Completas:** Deben cubrir la mayor cantidad de código.
- **Repetibles o Reutilizables:** No se deben crear pruebas que sólo puedan ser ejecutadas una sola vez.
- **Independientes:** La ejecución de una prueba no debe afectar a la ejecución de otra.
- Profesionales: Las pruebas deben ser consideradas igual que el código, con la misma profesionalidad, documentación, etc.

Las ventajas de realizar este tipo de testeo serían:

- Facilitan la refactorización, puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.
- Simplifican la integración de las partes al aumentar el grado de seguridad del correcto funcionamiento del código.
- Obligan a modularizar mejor el código

A continuación mostraremos una parte de código de testeo de uno de los diversos gateways de los que dispone nuestro proyecto:



```python
__author__ = 'dcortes'

import pytest

from dexma_drivers.mongo_connector import MongoConnector

from sentilo_domain.gateways.gateway_user_mongo import GatewayUserMongo


@pytest.mark.gateway
class TestGatewayUserMongo():

    _collection = "test_user"

    def _get_config(self):
        return {
                "host": "192.168.33.10",
                "port": 27017,
                "user": "dexcell",
                "password": "runTestsHere",
                "bd": "test_sentilo",
        }

    def setup_method(self, method):
        config = self._get_config()
        self.manager = MongoConnector(config)
        self.gateway = GatewayUserMongo(self.manager, self._collection)

    def teardown_method(self, method):
        self.manager._db.test_user.drop()

    @pytest.mark.unitary
    def test_basic_get(self):
        raw_data = {u"_id": u"1",
                    u"dep_id": u"1",
                    u"endpoint": u"1",
                    u"identifier": u"1",
                    u"provider": u"1",
                    u"auth_token": u"1"}
        self.manager.insert_object(self._collection, raw_data)
        entity = self.gateway.get("1")
        assert(entity == raw_data)
```

Ilustración 25 código usando py.test para testear

En la ilustración 25 podemos observar un trozo de código con varias cosas interesantes:

- la primera es nuestra configuración de mongo. Como podemos ver el host es una ip interna 192.168.33.10, esto es debido al uso de Vagrant para lanzar un entorno en cuestión de segundos cosa ideal para realizar tests de manera limpia
- La segunda son los métodos setup_method y teardown_method, estos dos métodos se lanzan cada vez que se lanza y finaliza un método de nuestra clase, de esta manera podemos realizar todo el código común en setup_method y usar el teardown para limpiar nuestra base de datos cada vez que finaliza la ejecución de un método. Con estas funciones

conseguimos lanzar cada uno de los tests de manera totalmente independiente y por tanto asegurando su veracidad.

- El tercer factor interesante que nos ofrece pytest son los tags, como podemos ver nuestro método test_basic_get se le ha asignado un tag de testeo unitario.

El resultado para el test de la clase que estamos viendo (test completo de todos los métodos que la componen no solo tst_basic_get) sería el siguiente:

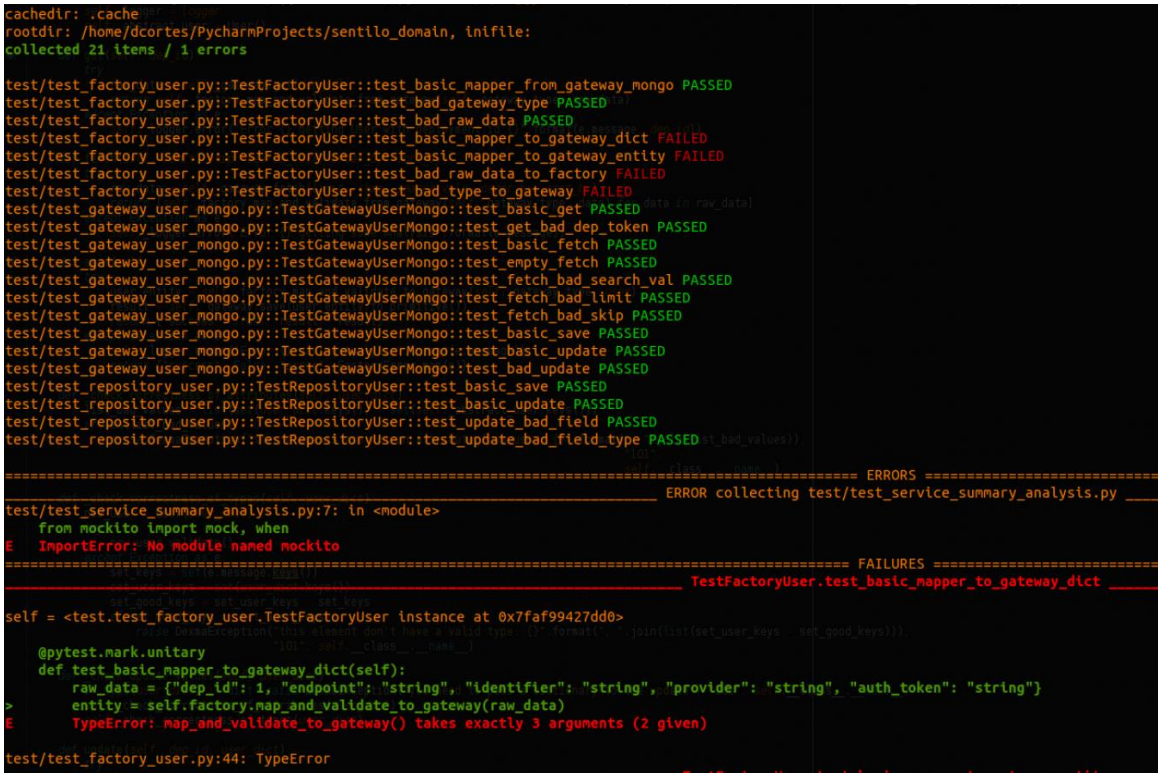


Ilustración 26 py.test en acción!

En esta imagen podemos observar el resultado en caso de fallos, como es el caso, a continuación el trozo de código donde se han producido estos.

8.2 Prueba de integración

Son aquellos tests que se realizan una vez aprobados los tests unitarios, su única función es la de probar todos los bloques unitarios que conforman un proceso conjuntamente para comprobar que el flujo y las conexiones entre bloques son correctos.

En el caso de nuestro proyecto para cada parte realizamos los tests de integración junto con los unitarios con py.test, en la ilustración anterior podemos apreciar testeos del repositorio los cuales son de integración ya que están usando el Factory y Gateway auténticos para testear todo el proceso de obtener una entidad.

8.3 Prueba de sistema

Estos son los tests dedicados a comprobar las necesidades externas de nuestro proyecto para poder adaptarse a un nuevo sistema, es decir todos esos elementos de la infraestructura como serian librerías externas, configuraciones, bases de datos…

Para este apartado lo que se hizo fue cargar todo el proyecto en una imagen de un sistema linux limpio, del mismo modelo que el servidor que luego contendría nuestro software, una vez incorporado se fueron viendo todas las necesidades y añadiendo todas estas en un fichero ansible para permitir automatizar todo el proceso de lanzar nuestra aplicación.

Como resultado obtuvimos varios ficheros .yml que contenían la sequencia a ser lanzada por ansible, para observar un ejemplo podemos observar el fichero que lanza nuestra aplicación web de Sentilo

```
---

- hosts:
    - market
  serial: 1
  vars:
    - app: sentilo_webapp
    - svn_url: https://subversion.assembla.com/svn/dexmarket.sentilo_export/

  tasks:
    - set_fact: svn_branch="trunk"

    - name: Deploy code
      subversion:
          executable: /usr/bin/svn
          repo: "{{ svn_url }}/{{ svn_branch }}/"
          dest: /opt/apps/{{ app }}
          revision: "{{ revision|default(omit) }}"
          username: "{{ svn_deploy.user }}"
          password: "{{ svn_deploy.password }}"
      register: deploy

    - name: Update requirements
      pip:
          requirements: /opt/apps/{{ app }}/requirements.txt
          virtualenv: /opt/venvs/{{ app }}
      register: dependencies

    - name: Restart uwsgi
      file: dest=/etc/uwsgi-emperor/vassals/{{ app }}.ini state=touch
      when: deploy.changed or dependencies.changed

--
```

Este fichero contiene la siguiente información

- Hosts: indica a que servidor atacar, en este caso market nombre especificado en un fichero de configuración donde se defiene la URL real del servidor
- Vars: permite crear variables a usar en el apartado tasks, en este caso observamos dos, app que hace referencia al nombre de la aplicación y svn_url que contiene la dirección de donde bajarse el código.
- Tasks: posee las tareas reales a ejecutar en el host especificado, cada tarea posee de dos valores que son name, nombre que se observará al lanzar el script en ansible, y un nombre de acción a ejecutar, en nuestro caso vemos dos que son subversión, encargado de bajar/actualizar el código y pip encargado de cargar todas las librerías Python de las que depende nuestro proyecto.

8.4 Prueba de aceptación

Pruebas realizadas por el usuario para determinar si la aplicación cumple las normas prefijadas.

En el caso de Dexma esta tarea recayó en un miembro del equipo de operaciones quien se dedico durante un par de días a ejecutar todas las acciones disponibles en las dos aplicaciones y comprobar el correcto funcionamiento de la importación y exportación.

8.5 Prueba de estrés

Esta prueba fue realizada simultáneamente a la de volumen, puesto que lo que se intentaba observar es si al lanzar muchos workers en una misma máquina podíamos llegar a consumir algún recurso de esta.

Las pruebas salieron mas que satisfactorias puesto que se comprobó que para ocho trabajadores en una máquina de ocho núcleos de CPU el sistema de workers era capaz de funcionar correctamente sin saturar ni la CPU ni consumir demasiada RAM.

Otro punto que se comprobó es que a partir de más workers que núcleos en una máquina la velocidad de consumo por workers no aumentaba prácticamente, y por tanto no era recomendable, sino que en caso de ser necesario lo mas óptimo era aumentar el número de máquinas.

8.6 Prueba de Volumen

se trata de verificar la funcionalidad de la aplicación tras soportar una vasta cantidad de datos.

Esta prueba se realizo manualmente insertando desde el Dispatcher una gran cantidad de exportaciones ha ser procesados, por supuesto el sistema no tuvo ningún problema en absorber estas peticiones ya que nuestro sistema de colas mantenía almacenadas las tareas hasta que algún worker quedara libre para ocuparse de la siguiente.

8.7 Alfa

las realiza el usuario final el ambiente del desarrollador quien notifica el comportamiento de este.

Estas son las pruebas de las que se ocupo la empresa solicitante de esta integración, quien a través de un ciclo de la mejora continua o retroalimentación fue depurando los detalles de la aplicación, sobretodo temas de visualización.

9 Análisis económico

Para realizar este análisis se ha tenido en cuenta todas las etapas desde el momento que se realizó la petición del acoplamiento por parte de Sentilo hasta su subida en producción por parte del equipo de Dexma.

Podemos separar el estudio en varias etapas, donde podemos encontrar profesionales tan diversos como:

- Los comerciales dedicados a realizar las primeras aproximaciones con Sentilo.
- El gestor del producto empleado a desarrollar los casos de uso (en nuestro caso llamadas “historias de usuario” de la metodología Scrum) y la definición de oportunidades.
- El director técnico encargado de realizar las primeras aproximaciones técnicas con el equipo técnico de Sentilo.
- El desarrollador ocupado de realizar el producto final.
- El administrador de sistemas encargado de crear los entornos de preproducción y producción para el funcionamiento de este.

9.1 Planificación proyecto

Para el planteamiento de este producto y así permitir una fácil observación de las etapas previstas y las duraciones de estas, se ha decidido usar una de las herramientas ya básicas para la gestión, **el diagrama de Gantt**.

El diagrama de Gantt es una herramienta gráfica que nos permite visualizar las duraciones de las etapas, las dependencias entre ellas y el camino crítico del proyecto facilitando asi la asignación de recursos.

En la página siguiente podemos observar nuestro diagrama Gantt conteniendo todas las fracciones del proyecto desde las negociaciones hasta su subida en producción.

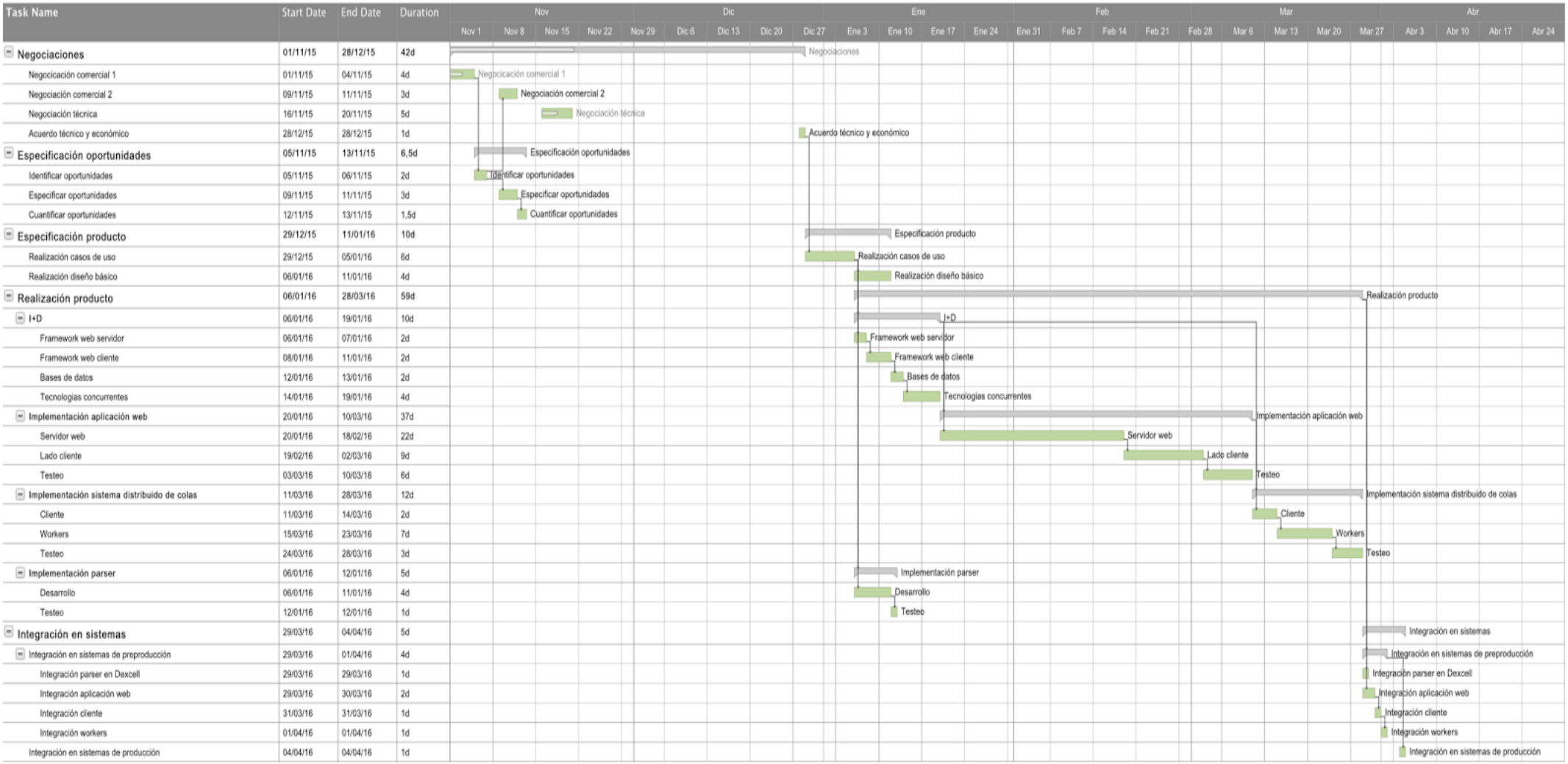


Ilustración 27 Diagrama Gantt

9.2 Estimación de costes

En este apartado se realizará una estimación aproximada del coste total de desarrollo de la aplicación.

Para ello hay que tener en cuenta un conjunto de factores desde los recursos humanos, hasta el material usado por este, el cual incluye hardware y el coste estructural.

9.2.1 Recursos humanos

Para la estimacion de sueldos, sabiendo que todos los miembros forman parte de Dexma se ha realizado la siguiente tabla:

Rol	Sueldo
Vendedor	20€/hora
Director técnico	35€/hora
Gestor del producto	30€/hora
Programador	20€/hora
Administrador de sistemas	25€/hora

Por tanto a través de la tabla de sueldos y el diagrama de Gantt observado anteriormente podemos hacer una aproximación muy certera del coste total del personal en este proyecto.

	Rol	horas	coste
Negociaciones			

Negociaciones comerciales 1	departamento comercial	16	320
Negociaciones comerciales 2	departamento comercial	12	240
Negociación técnica	Director técnico	12	420
Especificación oportunidades	Gestor del producto		
Identificación oportunidades	Gestor del producto	16	480
Especificación oportunidades	Gestor del producto	24	720
Cuantificar oportunidades	Gestor del producto	12	360
Especificación producto	Gestor del producto		
Realización casos de uso	Gestor del producto	48	1440
Realización diseño básico	Gestor del producto	32	960
Realización producto	Programador		
I+D	Programador		
Framework web servidor	Programador	16	320
Framework web cliente	Programador	16	320
Bases de datos	Programador	16	320
Tecnologias concurrentes	Programador	32	
Implementación aplicación web	Programador		
Servidor web	Programador	176	3520
Lado cliente	Programador	72	1440
Testeo	Programador	48	960

Implementación sistema distribuido de colas	Programador		
Cliente	Programador	16	320
workers	Programador	56	1120
testeo	Programador	24	480
Implementación parser	Programador		
Desarrollo	Programador	32	640
Testeo	Programador	8	160
Integración en sistemas	Administrador de sistemas		
Integración en sistemas de preproducción	Administrador de sistemas		
Integración parser en Dexcell	Administrador de sistemas	8	200
Integración aplicación web	Administrador de sistemas	16	400
Integración cliente	Administrador de sistemas	8	200
Integración workers	Administrador de sistemas	8	200
Integración en sistemas de producción	Administrador de sistemas	1	20
Total		829	14760

9.2.2 Coste estructural

El coste estructural ha sido calculado según los diversos gastos mensuales que tiene Dexma, que son:

- Alquiler de oficinas.
- Limpieza
- Seguranza
- Material de oficina
- Gastos varios (agua, supermercados, suministros)
- Teléfono fijo
- Teléfonos móviles
- Comisiones bancarias
- Otros costes de ventas

Al comprobar el gasto total de todos estos puntos nos da que Dexma gasta de media unos diez mil euros mensuales en estructura, estos se tienen que repartir entre los veinte cuatro miembros del equipo por tanto se puede decir que los gastos estructurales mensuales por persona que Dexma debe soportar son de 417 euros.

Si contamos que cada mes suponen veinte días laborales de una media de ocho horas diarias por empleado, tenemos que en total un mes dispone entre todos los empleados de 3840 horas productivas (8 horas * 20 días * 24 empleados), sabiendo esto y que la duración total del proyecto ha sido de cinco meses de las cuales se han destinado un total de 829 horas, podemos obtener fácilmente el gasto total por costes con una regla de tres.

- 3.840 horas / mes * 5 meses = 19.200 horas
- 10.000 euros / mes * 5 meses = 50.000 euros
- 829 horas * 50.000 euros / 19.200 horas = 2.159 euros

Por tanto podemos comprobar que el coste por estructura total del proyecto ronda los 2.159 euros.

9.2.3 Gastos imprevistos

Son considerados costes imprevistos un fallo en una torre de un pc o bajas temporales de algún miembro del equipo durante el tiempo de realización del proyecto.

- Para el caso del ordenador se estimará un coste de reparación de unos 200€.
- Para los casos de bajas vamos a asignar un porcentaje de un 5% del total de tiempo donde algún miembro debe ausentarse por motivos personales, de enfermedad o simplemente por vacaciones, lo cual supone un incremento en los gastos de 855€.

9.2.4 Coste total

Puntos	€
recursos humanos	17.100
coste estructurales	2.159
Gastos imprevistos	1.155
Total	20.414

10 Conclusiones

10.1 Objetivos y resultados

En este apartado valoraremos los resultados obtenidos, basándonos en los objetivos planteados en un inicio.

El planteamiento inicial del proyecto tenía los siguientes objetivos para Dexma:

- Proporcionar una interfaz de visualización específica para consumos energéticos a ayuntamientos que ya tienen datos en Sentilo

- Usar DEXCell como plataforma de recolección y normalización de datos provenientes de equipos de medida heterogéneos para enviar de forma homogénea datos a Sentilo

Todos estos objetivos han sido cumplidos correctamente gracias al cumplimiento de los objetivos técnicos que eran los siguientes:

- Desarrollo de una aplicación que permita al usuario configurar la exportación de datos energéticos de Dexcell a Sentilo a través de los dispositivos de Dexcell.

- Desarrollar un concentrador virtual en la plataforma de Dexcell que realice la tarea de importar los datos desde Sentilo a Dexcell de manera automática ocultando al usuario todo el sistema de suscripciones y su complejidad.

Todos estos objetivos se han cumplido de manera satisfactoria, ya que el aplicativo y el concentrador virtual realizados y integrados en Dexcell cumplen todas las expectativas que se propusieron, tanto a nivel funcional para el usuario, como a nivel técnico.

La integración bidireccional permite a los usuarios de Sentilo usar las funcionalidades de análisis de consumos energéticos que proporciona Dexcell y por otro lado permite a los usuarios de Dexcell compartir los datos de consumos energéticos con la iniciativa pública de IoT para que estos datos ayuden a entender el gasto energético en las ciudades.

Se ha conseguido la escalabilidad horizontal del sistema tanto para la importación como para la exportación gracias al uso de tecnologías como UWSGi + NGINX o Celery.

Por el lado de la importación ha sido el uso de UWSGI y NGINX puesto que gracias al sistema de suscripciones de Sentilo, nuestro bridge funciona como una API y por tanto se puede usar NGINX como load balancer para añadir nuevos servidores que traten las suscripciones y UWSGI te permite modular el nivel de paralelismo a nivel de servidor para elegir cuantos procesos asignamos internamente a la recepción de datos.

Por otro lado gracias a Celery hemos podido paralelizar de manera simple y efectiva el exportador permitiendo crear tantos trabajadores como se requieran para enviar los datos a Sentilo.

10.2 Conclusiones personales

En este proyecto a parte de profundizar conocimientos en Python o técnologias que ya solemos usar he tenido la oportunidad de usar Celery junto con Redis y luego RabbitMQ para realizar mi primera aplicación distribuida, donde he podido comprobar las grandes ventajas que implica este tipo de sistema pero a la vez he podido comprobar que esto también requiere un sistema más robusto y mas bien organizado para poder mantenerlo.

Otro punto en el que este proyecto me a ayudado a sido en lo referente al diseño, hasta ahora al siempre realizar aplicaciones de un solo proyecto nunca me había encontrado en una situación donde tuviera tanto código duplicado entre todas las partes del aplicativo y por eso he creído interesante profundizar en el diseño y aprender nuevas técnicas que permitan eliminar significativamente la duplicación de código. Gracias a esto he sido capaz de conocer la filosofía del DDD, entender sus grandes ventajas y también observar que el modo de realizar mis proyectos hasta ahora era bastante ineficaz entre otras cosas para futuras mejoras o incluso para el entendimiento de las responsabilidades de cada módulo.

Por último el hecho de haber usado Celery y poseer un aplicativo que podía consumir bastantes recursos de la máquina me ha hecho por primera vez realizar algunos tests de carga para observar los límites de nuestro sistema.

11 Propuestas de mejora

En este apartado comentaré brevemente algunos elementos del aplicativo que se podrían mejorar y el cómo.

11.1 Loggers centralizados

En un primer momento lo que se pudo comprobar al disponer de varios workers en varios servidores es como se ampliaba el trabajo de encontrar errores en los logs. Cada vez que algún dispositivo daba problemas en el envío de datos a Sentilo debía proceder a buscar en los logs de cada servidor hasta encontrar el log que enviaba la excepción y desde allí procedía a resolver el error.

Por tanto una propuesta de mejora muy importante en aplicaciones distribuidas como está es el poder disponer de un log centralizado que permita una visualización rápida de donde se ha producido el error, y poder también seguir el trazo de ese dato incorrecto desde su entrada hasta el error, para esto se ha empezado a mirar la opción de usar como solución ElastichSearch con fluentd y Kibana, para ver cómo funcionan podemos ver la imagen siguiente:

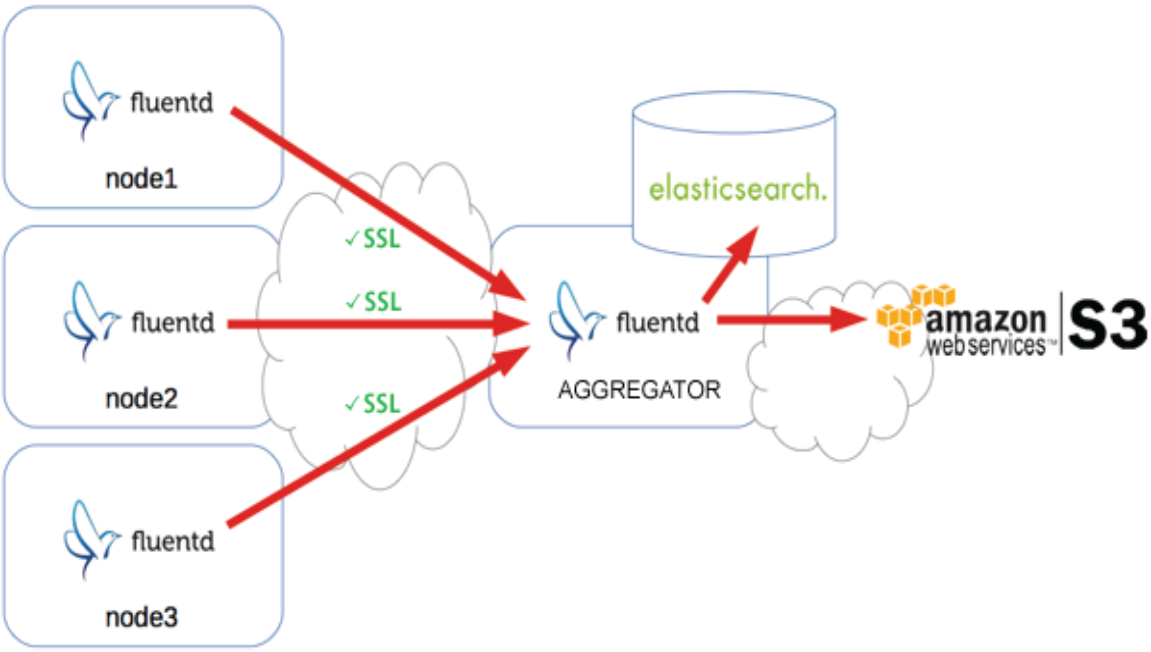


Ilustración 28 Diagrama fluentd con elasticsearch

Como se puede ver en la ilustración anterior se dispondría de una instancia principal de fluentd en el mismo servidor que elasticsearch que se ocuparía de agregar todos los logs obtenidos por las diferentes instancias de fluentd de los diversos servidores donde nuestro aplicativo corre y seguidamente almacenarlos en Elasticsearch una base de datos NoSQL construido por encima de Apache Lucene.

Una vez almacenados Los datos solo debemos configurar Kibana para visualizar estos de la manera que más nos convenga, tanto si esta es a traves de gráficos o de tablas.

11.2 BDD - behaviour Driven Development

Se ha considerado aplicar la filosofía BDD al proyecto para hacerlo más robusto, es decir, el uso de herramientas como cucumber para permitir una automatización del "ubiquitous language" a través de sus escenarios. Esto permite a través de un lenguaje natural como es el inglés exponer los diversos procesos que debe realizar el dominio del aplicativo y testearlos siguiendo los pasos expuestos en los escenarios.

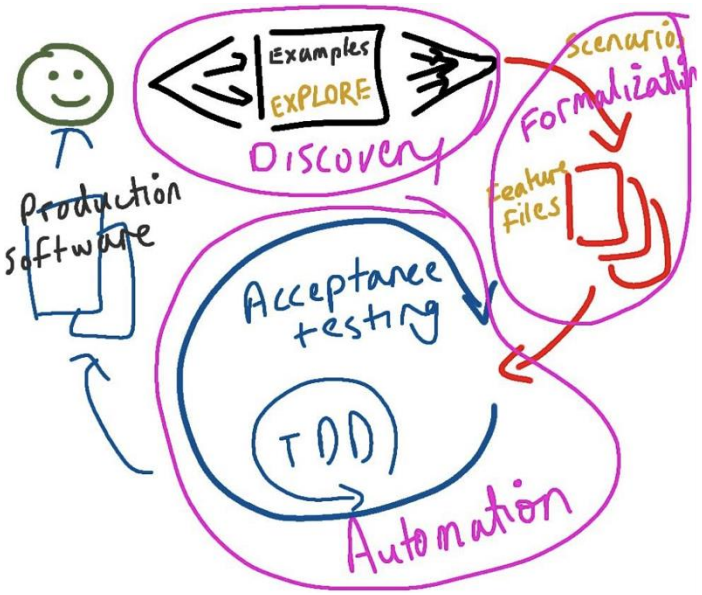


Ilustración 29 Explicación BDD Cucumber

Con esto no solo se propone hacer más robusto el domino del aplicativo, sino tambien a través de frameworks permitir testear la interfaz usando los escenarios para replicar las acciones que un usuario normal haría a diario.

11.3 Migración sistema almacenaje

Por el modelo descrito de entidades, se ha observado que seria mucho mas eficiente para almacenar los usuarios, componentes y sensores un modelo orientado a grafos como podría ser Neo4J, puesto que permite un rápido acceso a elementos continuos, por ejemplo todos los sensores relacionados a un componente, o todos los componentes a un usuario.
En el caso de Mongo al tener en un mismo documento todos los sensores relacionados a un componente, podríamos encontrarnos en la situación en que por cantidad de sensores el documento se tuviera que partir dificultando su acceso. Otro problema que evitamos con este cambio es la búsqueda de todos los componentes de un mismo deployment, puesto que Mongo no está orientado a joins, y esta relación necesita de una join entre dos colecciones, mientras que en un modelo por grafos está obtención sería directa.

11.4 PIP interno

Hasta hace poco en Dexma no contábamos con un servidor PIP propio, es decir, no disponíamos de un gestor de paquetes internos para python, haciendo imposible disponer de librerías privadas, pero este hecho a cambiado recientemente, por tanto una mejora posible sería empaquetar el dominio del aplicativo en una librería externa para poder hacerlo reusable para cualquier otro tipo de aplicación que contactara con la API de Sentilo.

12 Apéndice

12.1 Instalación de la aplicación

Como hemos mencionado anteriormente la aplicación de Sentilo ha sido desarrollada con la intención que forme parte del Market Apps de DEXcell y por tanto hemos de tener en cuenta los métodos de conexión y comunicación de este para nuestro desarrollo.

Primero de todo hemos de entender como funciona internamente el sistema de autentificación, que en el caso de DEXcell se basa en OAUTH.
En un primer momento y dentro de DEXcell como un usuario administrador de una cuenta vamos a nuestra pantalla de gestión de cuentas como podemos ver en la figura siguiente.

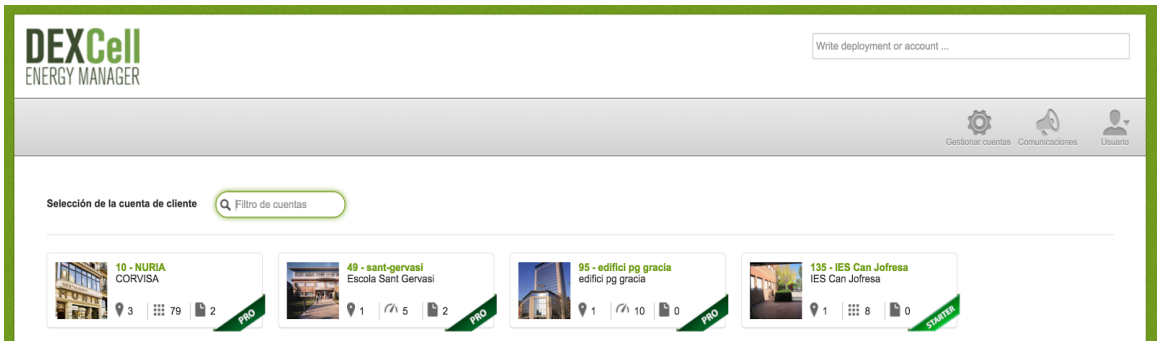


Ilustración 30 Selección gestión cuenta

Seguidamente seleccionamos Aplicaciones de Market como se observa en la siguiente figura.

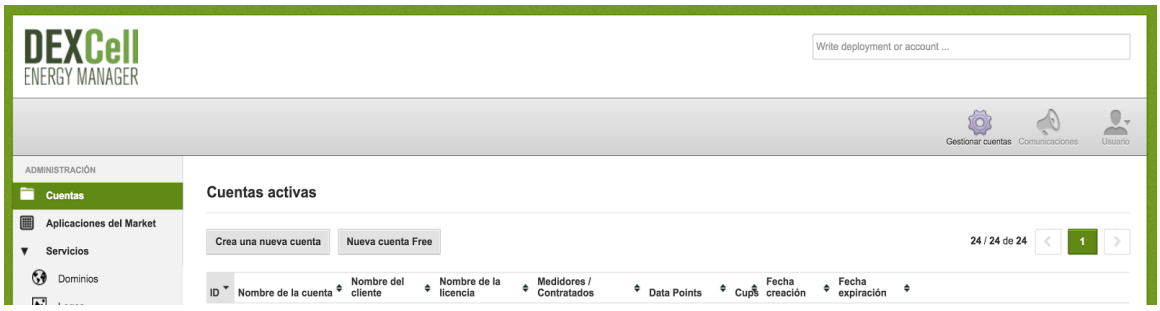


Ilustración 31 Selección aplicaciones market

Para finalizar Nueva Market App

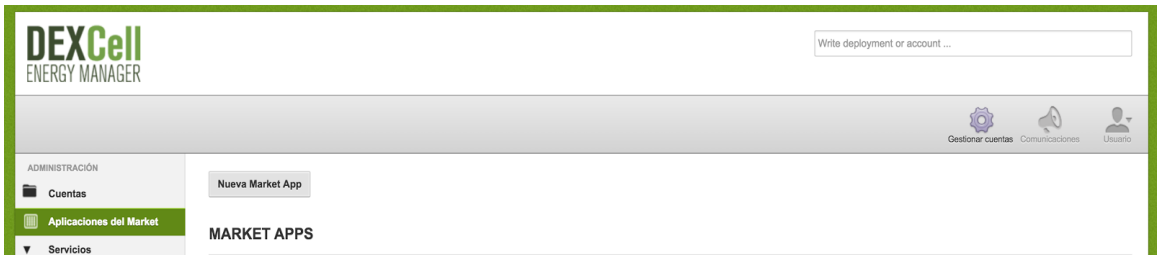


Ilustración 32 Selección nueva market app

Ahora debemos en la página que se nos muestra asignar un nombre y descripción a nuestra aplicación, y más importante para nosotros como desarrolladores, la dirección en donde podemos encontrar nuestra aplicación (en desarrollo usaremos 0.0.0.0) con todas sus vistas asignadas y en qué parte de Dexcell serán incorporadas. En el caso de nuestra aplicación solo ara falta una vista que hemos llamado Sentilo Export y redirigida a la raíz de nuestro dominio http://sentilo-export.enerapp.com/.

Ilustración 33 Configuración creación app

Por último tenemos que asignar los permisos que tendrán nuestros usuarios a través de esta aplicación para cada recurso proporcionado por Dexcell (localizaciones, dispositivos…) estos permisos se pueden observar en la siguiente figura, que como hemos explicado anteriormente serán aplicados a los tokens asignados a cada usuario para la aplicación. En el caso de Sentilo Export todos los permisos los dejaremos en lectura, puesto que en donde escribimos es en la aplicación de Sentilo y por tanto la API de dexma solo es usada para leer los valores.

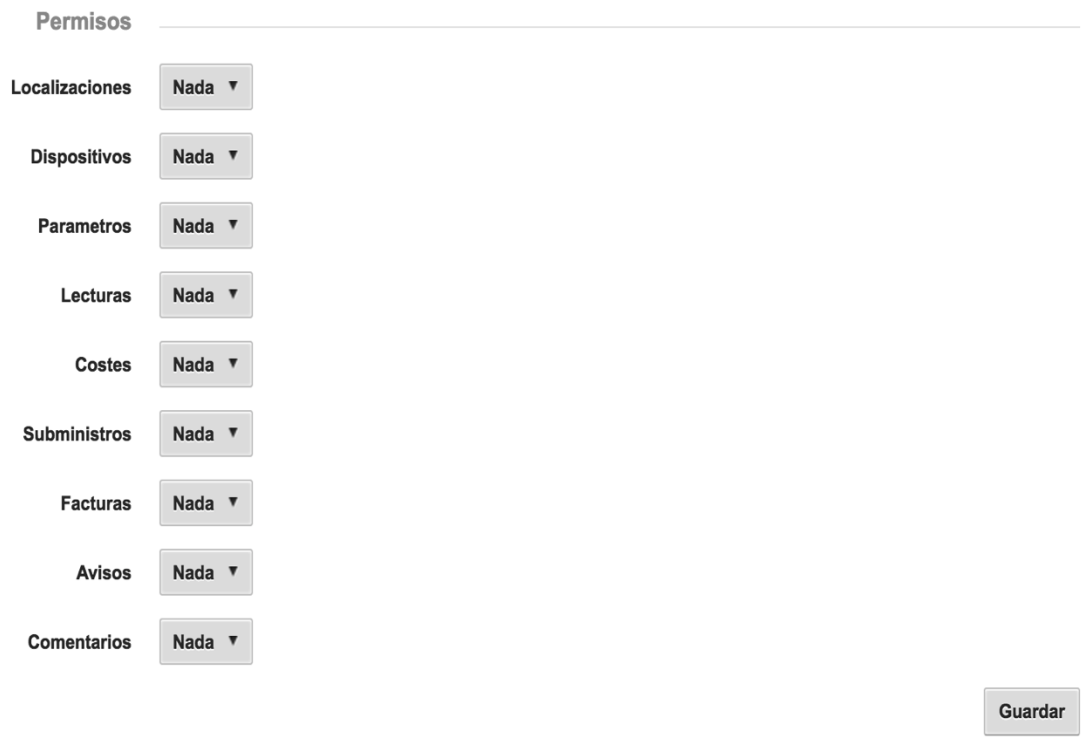


Ilustración 34 Confiugración permisos app

Una vez finalizado, podemos guardar esta aplicación y en el menú general del Market Apps deberíamos ver una nueva entrada conteniendo el nombre de la aplicación, su id, secret y visibilidad.

El id y secret son valores muy importantes que usaremos a continuación para la autorización.

La visibilidad hace referencia a quién puede ver la aplicación dentro de Dexcell, en un primer momento siempre tendrá visibilidad privada, es decir, sólo será visible para el usuario que ha creado el aplicativo, una vez este usuario, en este caso nosotros, decida hacer pública su aplicación para todos los usuarios de Dexcell deberá hacer una petición a Dexma. La empresa después de realizar un control de calidad y aprobar el contenido actualizará el estado a público.

Ahora que ya tenemos creada la aplicación en DEXcell podemos ir al market apps para observar nuestra nueva aplicación e instalarla cuando queramos!

Como podemos ver en este conjunto de pasos para llegar a instalar la aplicación antes hemos de conceder un conjunto de permisos a la aplicación y es en este momento en el que damos por segunda vez click a un botón instalar es cuando dexcell llama a la aplicación a través de la url especificada en el sector de URL's, instalación y empieza el proceso oauth.

En esta primera interacción entre DEXcell y el aplicativo es donde se crea el token permanente para el usuario para esta aplicación.

Este proceso sucede gracias a que DEXcell transmite a la aplicación una serie de datos como el token temporal y el identificador del deployment dentro de DEXcell. La aplicación en cuestión, en nuestro caso Sentilo iniciará una petición oauth usando para esta el id y secret de la aplicación propia y el token temporal generado específicamente para esta llamada. Esta petición se realizará a través de la API de dexcell usando la llamada http://api.dexcell.com/v3/oauth/access-token?temp_token=TEMP_TOKEN&secret=SECRET&app_id=APP_ID que nos devolvera el token permanente asociado a este usuario para esta aplicación concreta.

Este token debe ser guardado de manera persistente puesto que será el usado por el usuario cada vez que quiera usar la aplicación, por tanto al ser un dato tan importante y que debe ser de rápido acceso para esto se decidió usar un Redis en el mismo servidor donde se encuentra la aplicación para agilizar el proceso lo máximo posible.

```python
def render_bad_page(error, callback):
    render_html = render_template("errors/general_error.html", error=error, callback=callback)
    return render_html


def execute_nice_token(dro, dep_id, callback, logger_name):
    temp_token = unicode(request.args.get("temp_token", None))
    perm_token = dro.get_oauth_token(temp_token)
    bd = current_app.config["managers"]["redis"]
    tok_model = token_model(bd)
    state = tok_model.save_token(logger_name, dep_id, unicode(perm_token))
    if state is False:
        return render_bad_page("Some problem configuring the application", callback)
    else:
        mongo = current_app.config["managers"]["mongo"]
        log_model = logsModel(mongo)
        log_model.create_log_collection(logger_name, dep_id)
        return redirect(callback)


@mod_setup.route('/')
def setup():
    """
    We use the temp_token from the URL to take the perm_token for the client
    After using the dep_token (it doesnt' change for the client) we save it on redis
    Ass this application save information in redis and we need to knwo how many deployments have it, every time one user
    register in the app he will be added into a list
    :return:
    """
    current_app.logger.info('installing virtual_app')
    try:
        callback = str(request.args.get("callback", "http://dexcell.com/"))
        DEXCELL_ENDPOINT = current_app.config['DEXCELL_ENDPOINT']
        ID = current_app.config['ID']
        SECRET = current_app.config['SECRET']
        LOGGER_NAME = current_app.config['NAME']
        dro = DRapi_oauth(DEXCELL_ENDPOINT, ID, SECRET, logger_name=LOGGER_NAME)
        dep_id = request.args.get("dep_id", None)
        if dep_id is not None:
            page_redirect = execute_nice_token(dro, dep_id, callback, LOGGER_NAME)
        else:
            page_redirect = render_bad_page("Deployment token null", callback)
    except:
        current_app.logger.exception('problem with setup!!!')
        page_redirect = render_bad_page("Unknow problem with setting the session", callback)
    return page_redirect
```

Ilustración 35 Código encargado de gestionar la creación y guardado del token

Arriba en la figura XX podemos observar como nuestra aplicación diseñada en Flask obtiene de los parametros pasados por url el elemento “temp_token” que hace referencia al token temporal del usuario, con este y con el ID y SECRET obtenidos por la configuración hacemos la llamada para obtener el token permanente, perm_token = dro.get_oauth_token(temp_token), si este no es nulo y por tanto contiene un valor correcto lo guardamos en redis a traves de la clase tok_model. Para finalizar en el caso de Sentilo debemos inicializar una “capped collection” en mongo para este usuario que será usada para guardar los últimos cien logs y asi poder mostrarle al usuario en qué estado han finalizado las últimas acciones realizadas por Sentilo Export internamente.

13 Referencias

Bases de datos

Seven databases in seven weeks O'reilly Eric Redmond and Jim R.Wilson

RabbitMq

https://www.rabbitmq.com/documentation.html

Celery

http://www.celeryproject.org/

MongoDB

https://www.mongodb.org/

Redis

http://redis.io/

redis cookbook o'reilly Tiago macedo and Fred Oliveira

Flask

http://flask.pocoo.org/

Vagrant

vagrant up and running O'reilly, Mitchell Hashimoto

https://www.vagrantup.com/

Jenkins

Jenkins the definitive guide o'reilly John Ferguson Smart

Ansible

http://www.ansible.com/

Scrum

http://proyectosagiles.org/que-es-scrum/

Pytest

http://pytest.org/latest/

Sensu

https://sensuapp.org/

NGINX uWSGI

https://www.digitalocean.com/community/tutorials/how-to-deploy-python-wsgi-applications-using-uwsgi-web-server-with-nginx